Treball Fi de Màster

Master's Degree in Automatic Control and Robotics

3D Pose Estimation Using Convolutional Neural Networks

Master Thesis

Autor: Antonio Rubio Romano
Director/s: Francesc Moreno Noguer
Michael Alejandro Villamizar Vergel

Convocatòria: October 2015

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Acknowledgements

I would like to thank in the first place my thesis advisor Francesc Moreno, who introduced me into the Computer Vision world, who gave me the opportunity to continue working at the Institut de Robòtica i Informàtica Industrial (IRI) developing this master thesis and who always came up with the best ideas. Special thanks to Michael Villamizar, who helped me when the code was full of bugs and results were not as good as expected, and who contributed with the implementation of the sliding window algorithm.

To Cecilio Angulo, Master coordinator, who helped me whenever I needed it.

To all the IRI partners that made me play football again, and to those whose stories I enjoyed listening during many lunch breaks.

To all the friends who visited me in Barcelona filling this city with good memories along these two years.

To my family, who always supported me.

Abstract

The present Master Thesis describes a new Pose Estimation method based on Convolutional Neural Networks (CNN). This method divides the three-dimensional space in several regions and, given an input image, returns the region where the camera is located.

The first step is to create synthetic images of the object simulating a camera located at different points around it. The CNN is pre-trained with these thousands of synthetic images of the object model.

Then, we compute the pose of the object in hundreds of real images, and apply transfer learning with these labeled real images over the existing CNN, in order to refine the weights of the neurons and improve the network behaviour against real input images.

Along with this *deep learning* approach, other techniques have been used trying to improve the quality of the results, such as the classical sliding window or a more recent class-generic object detector called objectness.

It is tested with a 2D-model in order to ease the labeling process of the real images.

This document outlines all the steps followed to create and test the method, and finally compares it against a state-of-the-art method at different scales and levels of blurring.

Contents

Chapter 1

Introduction

Robust camera localization is one of the most relevant problems tackled by Computer Vision, and a fundamental part of the operations carried out by many different robots, such as precise manipulators or autonomous vehicles.

Despite being a largely studied topic, finding a fast and robust solution still represents a challenge nowadays, and there exist many different approaches. Some of them propose to localize the robot by equipping it with multiple sensors, such as lasers or stereo cameras, and then applying data fusion techniques. Whereas these systems improve the precision of the localization, they also increase the computational cost, which is a critical aspect in some applications (for instance, in tasks where small robots, e.g. aerial robots, are commonly used). Other approaches, such as Vicon [33] and similars, present excellent results based on infrared cameras and high-frequency systems, but nevertheless are limited to indoor environments, where lighting conditions are under control.

Opposite to these multi-sensor approaches, we present an efficient system based exclusively on a monocular camera and a pre-trained Convolutional Neural Network (CNN). No temporal constraints are taken into account that can reduce the region of the space where the camera is pointing at. This makes the resulting pose estimation robust to issues such as drifting, occlusions of the object or sudden camera motions, while making the problem significantly more complex.

1.1 Brief description of the proposed method

The proposed method takes a singular input image of the object (with whose model we previously trained a CNN) and estimates the camera pose, as shown in Fig. 1.1. We talk about camera pose (and not only position) because, although the network only returns the region of the space where the camera is located and not its orientation, the only cases where we obtain a valid region of the space for the camera position are when the object appears on

the image. And that means that we have at least an approximation of the orientation of the camera, since from every region in which we divided the space the object can only be seen within a certain range of camera orientations.

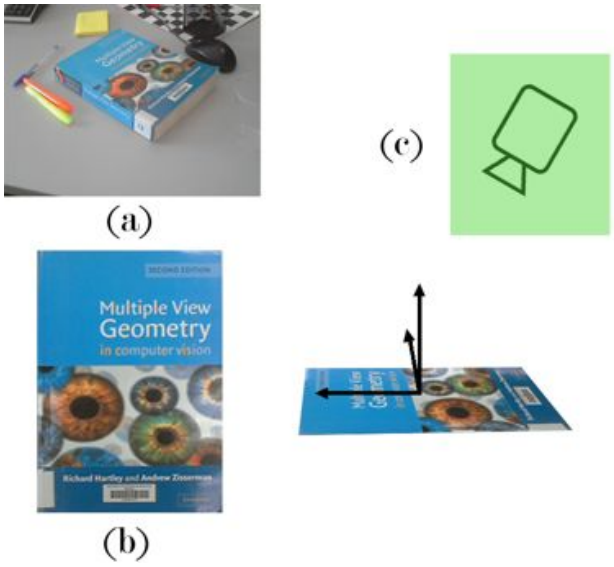

Figure 1.1: Problem definition: given an input image (a) and a known textured model of the object (b) with which we previously trained a CNN, the problem is to estimate the pose of the camera within a certain region (in green) that captured the image with respect to the object (c).

This method requires some offline work in order to be applied over an input image. The following is a list of the steps which are needed in order to obtain good results[1].

1. **Generation of synthetic images:** based on the model object, we take thousands of camera poses and create images simulating the object seen from those camera poses over a random textured background.
2. **Network training:** with the synthetic information created in the previous step, we train a specific convolutional neural network to classify input images into nine different classes estimating the camera position.
3. **Real images labeling:** here we compute the camera poses of hundreds of real images in order to label them for validating the method. Some of these images will be used for domain adaptation.
4. **Re-training: domain adaptation:** choosing to train images similar to the expected inputs yields better results. In this step, we re-train the network from step 2 with a few real images in order to tune the weights of the neurons adapting them to produce a better classification of the future real inputs.

Once this pipeline has been correctly followed, the method is ready to receive an input image and classify the position of the camera in one of the space regions that we will describe in 3.3.

[1]Steps 3 and 4 are optional, but strongly recommended when input images are real.

To summarize, we can narrow down the presented work in the following objective and contributions:

Objective of the project: The final goal of this master thesis is to develop a *deep learning*-based pose estimation method. We will choose an object, test several possible variants of the proposed method with it and then evaluate the results obtained with different metrics.

Contributions: We create a framework for estimating the pose of the camera with respect to an object within a certain region in a fast way, and prove it robust to a certain level of blurring of the input images. We perform several tests and analyze the results in order to comprehend under which circumstances the method offers a better performance.

1.2 Note about the model

Without loss of generality, the selected object used in this work is two-dimensional (a book cover). The main reason why this kind of object has been chosen is the fact that real images will need to be labeled manually, and for flat rectangular objects this can be done with an acceptable precision by simply clicking on the corners of the object in the image, as will later be explained in 3.2. Nevertheless, the extension of the method for 3D objects is direct, since the synthetic views of the object for different camera poses can be generated automatically with software solutions such as Blender [4]. Labeling of real images can be a harder problem when dealing with three-dimensional objects, but can also be solved by many PnP methods like [18].

That means that, having a correctly labeled dataset of synthetic and real images, the application of the method is always possible for three-dimensional objects.

The chosen book cover is that of the book *Multiple View Geometry in computer vision* [14], shown in Fig. 1.2, whose size is $246 \times 173mm$.

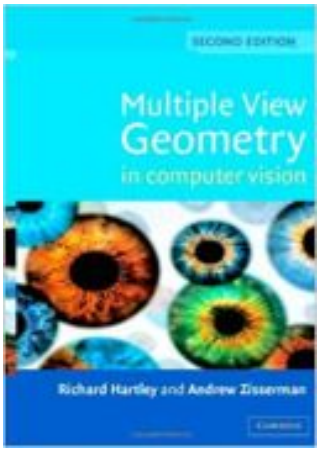

Figure 1.2: Cover of the book used to test the method.

1.3 Structure of the report

This report is structured as follows:

- Chapter 2 presents some related work on the fields of pose estimation and *deep learning*.
- In Chapter 3 we describe the process of creating the images used to train and test the method. Firstly, the creation of synthetic images is described, followed by the pose estimation of a set of real images and ending with the description of the labels given to each image based on its camera position.
- Chapter 4 details the convolutional neural network used by the method. It begins describing the structure of the network, then its training process and finally the implementation details.
- Then, Chapter 4.5 adds some variations of the method applying two windowing approaches.
- Chapter 5 explains the metrics used to evaluate the method and shows some of the results obtained (complete results are listed in Appendix A). It also shows the comparison against a state-of-the-art pose estimation method.
- Finally, we discuss the outcome of the project and set some future goals in Chapter 6.
- For the completeness of the results chapter, we exhaustively add all the results in Appendix A.

Chapter 2

Related work

The 3D pose estimation problem has been basically tackled from two perspectives in computer vision: either purely geometric feature-based methods or appearance-based machine learning approches.

Geometric methods are based on the extraction of 2D features in the image and the search of 2D-3D correspondences between the image and the three-dimensional model of the object. PnP algorithms such as the EPnP [22, 18, 25, 8] are applied later to geometrically constrain the solution. Outlier rejection strategies such as RANSAC can then be used to speed up the matching process. While providing very accurate results, all of these methods require very high quality models and input images since they rely on a good feature extraction.

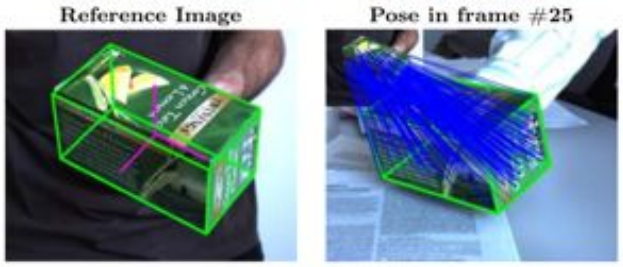

Figure 2.1: Reprojection of the 3D pose of an object computed from 2D-3D correspondences (from [18])

In contrast, similar to what we do in this work, machine learning techniques use discrete locations in the pose manifold to label a set of training images, and then compare the appearance of the input image to this pose-labeled set, without needing to perform a 3D reconstruction of the model [23, 34]. In order to model the spatial relationships between local features, some approaches use one single detector for all poses [15, 19, 29] and some combine various pose-specific detectors [24, 30]. Another option is to tie image features with poses in the training process and have them vote in the pose space [12] (see Fig. 2.2).

Appearance-based methods are in general more robust against image degradations than

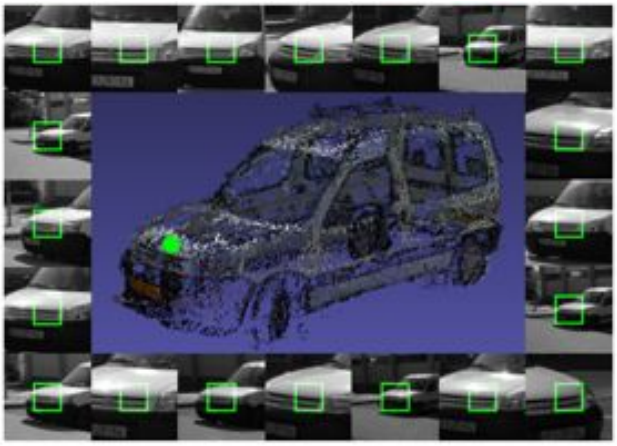

Figure 2.2: Different 2D appearances for a specific 3D point (from [12])

feature-based methods since they are not as sensitive to the exact localization of the features in the image. On the other hand, global methods are usually less precise since they depend on the granularity between the classes in which the pose space is divided, and present a worse behavior against occlusions. Some methods like [26] combine the advantages of the two approaches, and furthermore, recent works use *deep learning* to extract local features [28, 9] or to predict similarity between image patches [36]. These methods are patch based, leaving the spatial aggregation to a postprocessing step, while our method directly provides the final result using the output of the network.

Recently, convolutional neural networks trained with backpropagation [17] have been shown to perform well on large-scale image classification problems [16], which gave origin to many works applying CNNs to different computer vision problems. Also, new network architectures predict pixel values, which allows to address depth estimation [6], semantic segmentation [20, 7, 11, 13], keypoint prediction [13] or edge detection [10] problems.

The application of CNNs to the pose estimation problem, however, is not very frequent. A recent publication [35] uses a convolutional network to compute descriptors of different 3D models. In our case, we propose an appearance-based method using CNNs for estimating the pose of the camera with respect to an object.

Chapter 3

Dataset creation

As previously stated, the proposed method requires a pre-trained CNN for estimating the camera position.

In order to perform a good training process, a high number of images is required. The cost of taking thousands of photographs of a real object and labeling each of them individually with the correct pose of the camera is prohibitive in absence of an automatic method for doing it. For that reason, using synthetic images and then applying domain adaptation with a smaller subset of real labeled images constitutes a much better alternative.

This chapter will describe the creation of two labeled datasets: an initial group of thousands of synthetic images for the first training of the CNN (Section 3.1), and a second smaller group of some hundreds of real images used for domain adaptation and validation of the method (Section 3.2).

Finally, a description of the 9 labels used to group the different images is given in Section 3.3.

3.1 Synthetic Image Generation

As previously stated, the first step of the method consists of generating thousands of synthetic labeled images from an object model.

This process implies three steps:

1. Defining the positions and parameters of the *virtual* camera that will take the images (Section 3.1.1)

2. Computing the homographies of the book cover seen with that virtual camera from those positions (Section 3.1.2) using the available model of the object
3. Adding random backgrounds to the generated images to make them more similar to the real ones (Section 3.1.3)

3.1.1 Camera positions and parameters

First of all, we define a set of points in the three-dimensional space equally distributed along the surface of a semisphere surrounding the upper part of the object, as shown in Fig. 3.1.

Figure 3.1: Semisphere of 3D points generated around the object.

To add scale invariance, spheres of different radii are used, and then replied along axis $\boldsymbol{z_w}$ (see Fig. 3.2). The set of all these points form the virtual positions of the camera that will generate the synthetic images. Considering where the real photographs of the object will be taken from, we decided to choose positions between $[-36.86, 36.86]$ in $\boldsymbol{x_w}$ and $\boldsymbol{y_w}$, and $[10.68, 92.40]$ in $\boldsymbol{z_w}$.

In addition, before generating those images via the homography matrix, we need to define the camera parameters. In this case, the parameters were extracted from the calibration of the camera used to take the real pictures of the database, and are shown in Table 3.1.

Focal length $\boldsymbol{f_x}$ (pixels)	206.0358
Focal length $\boldsymbol{f_y}$ (pixels)	205.1238
Principal point (u_0, v_0) (pixels)	(80.0534, 49.8627)

Table 3.1: Virtual camera parameters.

Hence, the camera intrinsic matrix $\boldsymbol{A}$ is:

$$\boldsymbol{A} = \begin{bmatrix} 206.0358 & 0 & 80.0534 \\ 0 & 205.1238 & 49.8627 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

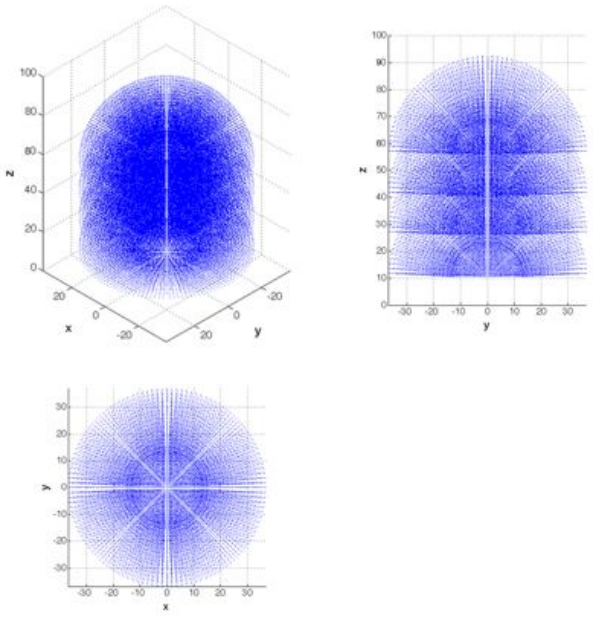

Figure 3.2: Points used to generate the synthetic images.

3.1.2 Homography computation

Next, the obtention of the homography matrix is described. First of all we need a model of the object. In our case, it is shown in Fig. 3.3. Note that this image is different from the one in Fig. 1.2. This one is a photograph taken from the real object, presenting some reflections (and being generally more similar to the real images that will later be used as inputs for the method), while the previous one was the *original* book cover image, not a picture taken from it. Using this model will improve the performance of the method against real input images.

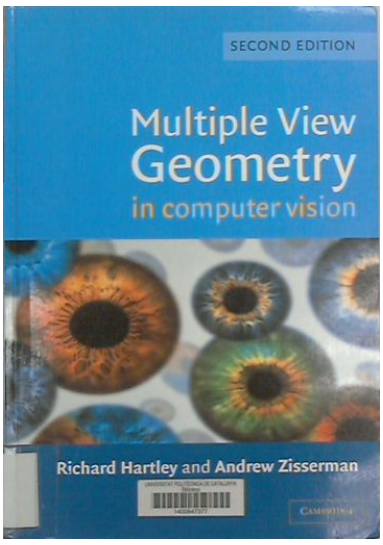

Figure 3.3: Model of the object.

First, we define the frame coordinates *world*, centered in O, and *camera*, centered in $\boldsymbol{c}$ as seen in Fig. 3.4. From now on, consider $\boldsymbol{v}^c$ as a vector $\boldsymbol{v}$ expressed in *camera* coordinates and, in absence of superindex, assume vector expressed in *world* (nevertheless, superindex $\boldsymbol{v}^w$ will be used when a specific distinction is useful).

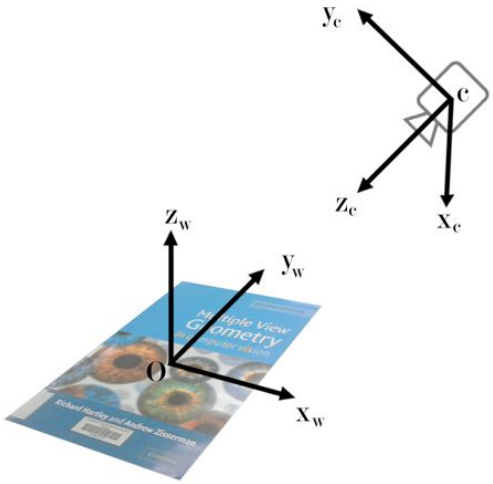

Figure 3.4: Coordinate frames.

Given a camera position, i.e. one of the points from Section 3.1.1, named $\boldsymbol{c}$ and expressed in *world* coordinates, and a target point the camera is looking at (that will later be defined in Chapter 5 as the center of our model with some noise, defined as $\boldsymbol{t}$, also in *world* coordinates), we can transform the object model to simulate the view from a camera located in the specified position $\boldsymbol{c}$ and looking at the specified target $\boldsymbol{t}$ using the correspondent homography matrix $\boldsymbol{H}$ defined in Eq. 3.2, being $\boldsymbol{A}$ the camera intrinsic matrix and $\boldsymbol{R}$ the rotation matrix described below.

$$\boldsymbol{H} = \boldsymbol{A} \cdot [\boldsymbol{R} \cdot \boldsymbol{c}] \tag{3.2}$$

We need to compute the rotation matrix $\boldsymbol{R}$ relating points in the *world* frame coordinate with points in the *camera* frame coordinate.

The columns of a rotation matrix $\boldsymbol{R}^\top$ relating the *world* frame with the *camera* frame would be the vectors of the *camera* frame expressed in *world*. We only know the camera position and the target point, and have to define the camera frame, formed by $\boldsymbol{x_C}$, $\boldsymbol{y_C}$ and $\boldsymbol{z_C}$. Hence, the matrix $\boldsymbol{R}$ we are looking for is shown in Eq. 3.3.

$$\boldsymbol{R} = \left[\boldsymbol{R}^\top\right]^{-1} = \begin{bmatrix} \boldsymbol{x_C}(1) & \boldsymbol{y_C}(1) & \boldsymbol{z_C}(1) \\ \boldsymbol{x_C}(2) & \boldsymbol{y_C}(2) & \boldsymbol{z_C}(2) \\ \boldsymbol{x_C}(3) & \boldsymbol{y_C}(3) & \boldsymbol{z_C}(3) \end{bmatrix}^{-1} \tag{3.3}$$

Since we want the camera to point towards the target, vector $\boldsymbol{z_C}$ is easy to compute following Eq. 3.4.

$$\boldsymbol{z_C} = \frac{\boldsymbol{t} - \boldsymbol{c}}{\|\boldsymbol{t} - \boldsymbol{c}\|} \tag{3.4}$$

For vectors $\boldsymbol{x_C}$ and $\boldsymbol{y_C}$ to form an orthogonal coordinate frame with $\boldsymbol{z_C}$, we will choose them included in the null space of $\boldsymbol{z_C}$. From the infinite possibilities, we choose to define them as stated below.

Given vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, base of the null space of $\boldsymbol{z_C}$, the vector $\boldsymbol{x_C}$ will be constructed as stated in Eq. 3.5.

$$\boldsymbol{x_C} = a \cdot \boldsymbol{u} + b \cdot \boldsymbol{v} \tag{3.5}$$

In order to solve the equation for a and b, we will impose the restriction of $\boldsymbol{x_C}$ being horizontal ($\boldsymbol{x_C}(3) = 0$). From that assumption we get the solutions shown in Eq. 3.6.

$$\begin{cases} a = 1,\ b = -\boldsymbol{u}(3)/\boldsymbol{v}(3), & \text{if } \boldsymbol{v}(3) \neq 0 \\ b = 1,\ a = -\boldsymbol{v}(3)/\boldsymbol{u}(3), & \text{if } \boldsymbol{v}(3) = 0 \wedge \boldsymbol{u}(3) \neq 0 \\ a = 1, b = 0, & \text{if } \boldsymbol{v}(3) = 0 \wedge \boldsymbol{u}(3) = 0 \end{cases} \tag{3.6}$$

With the values for a and b, we obtain $\boldsymbol{x_C}$ and then normalize it ($\boldsymbol{x_C} = \boldsymbol{x_C}/\|\boldsymbol{x_C}\|$) before obtaining the remaining vector of the frame with Eq. 3.7

$$\boldsymbol{y_C} = \boldsymbol{z_C} \times \boldsymbol{x_C} \tag{3.7}$$

Finally, we force $\boldsymbol{y_C}$ to point upwards (if $\boldsymbol{y_c}(3)$ is negative, we change its sign) in order to obtain an image that is not upside down, similar to the real images.

Now, for a given camera with intrinsic matrix A located at a point $\boldsymbol{c}$, and for any 3D point in space $\boldsymbol{p}$, from Eq. 3.2 we can obtain its 2D coordinates on the image plane seen by that camera, $\boldsymbol{p^c}$, with Eq. 3.8.

$$\begin{gathered} \boldsymbol{p^c} = \boldsymbol{A} \cdot \boldsymbol{R} \cdot \boldsymbol{p^w} + \boldsymbol{c} \\ \boldsymbol{p^c} = \boldsymbol{p^c}/\boldsymbol{p^c}(3) \end{gathered} \tag{3.8}$$

This means that, since we know the positions of all the points of the book cover, applying Eq. 3.8 yields as a result their positions on the image plane. By applying an interpolation algorithm we obtain a decent simulation of a picture of the object taken with the specified camera position ($\boldsymbol{c}$) and parameters ($\boldsymbol{A}$). This part of the process is illustrated in Fig. 3.5.

In Fig. 3.6 we show some examples of the images obtained by this process.

3.1.3 Background addition

Images obtained from previous section (3.1.2) are represented always over a white background (as seen in Fig. 3.6), but usually real images will present very different textured backgrounds.

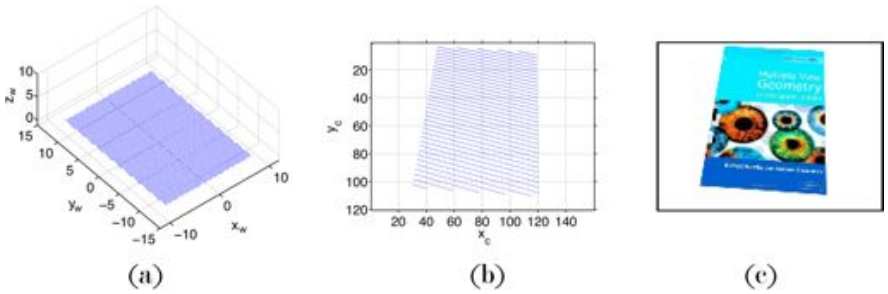

Figure 3.5: Points of the original object model in centimeters in *world* coordinates (a), same points on the image plane, i.e. expressed in pixels in *camera* coordinates (b) and resulting image (c).

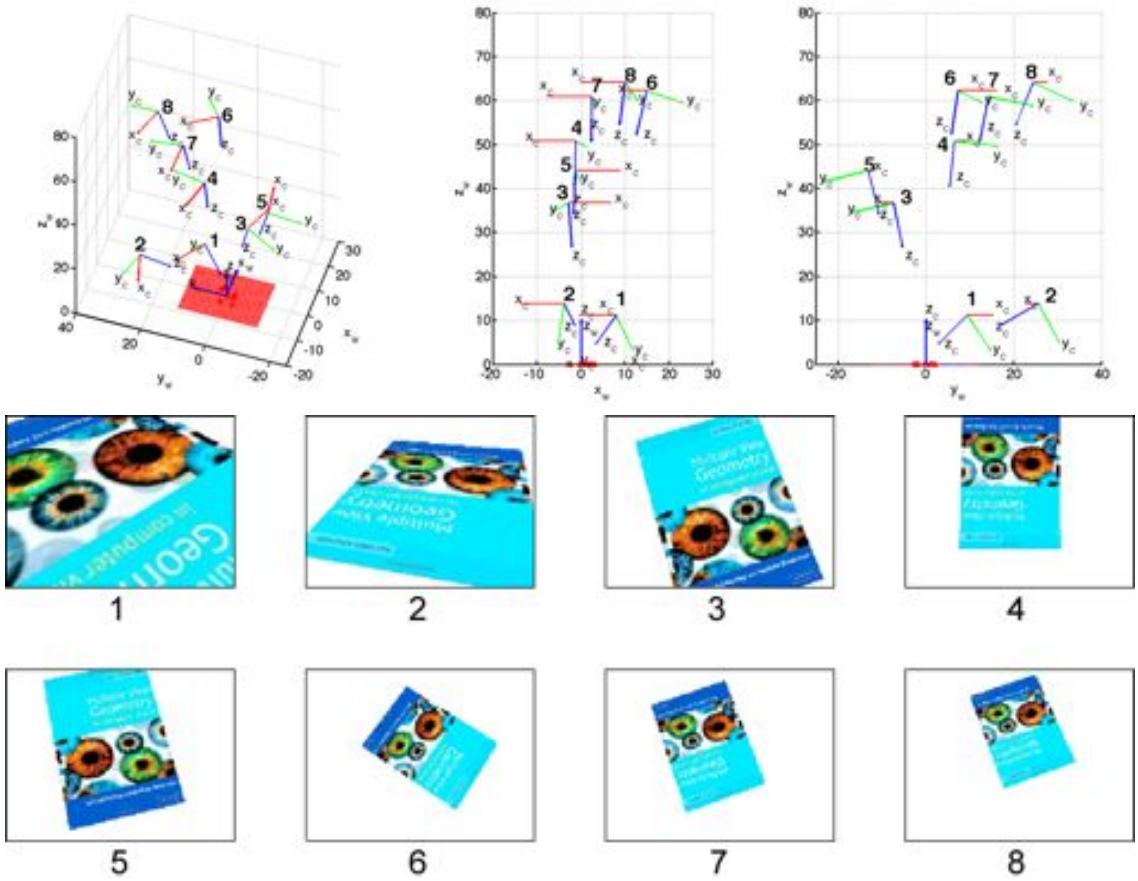

Figure 3.6: Eight examples of the images obtained with the homography matrix (bottom) computed for eight different camera positions (coordinates shown in top image).

Training with this kind of images will lead the CNN to learn the white background as part of the desired object, so feeding it afterwards with a picture of the real object over a textured background will produce wrong results. In order to avoid this problem, different backgrounds were added to the previous images.

We first decided to add random textured backgrounds to the images, extracted from the ETHZ Synthesizability dataset [5], as seen in Fig. 3.7.

Figure 3.7: Synthetically generated images with textured backgrounds from ETHZ Synthesizability dataset.

Pursuing better results, these textures were later substituted by real pictures of the background where our real object will lie. Therefore, the backgrounds used for making the synthetic images look more like the future input images are those shown in Fig. 3.8.

Figure 3.8: Pictures of the background where the real object will lie (left) and synthetically generated images with those pictures as background (right).

3.2 Real images labeling

Along with the synthetic images mentioned in Section 3.1, real images are needed in order to adapt the network to work with real input images.

The first step is to estimate the homography $\mathbf{H}$ that maps the plane defined by the book cover into the image plane. Assuming that the book cover lies on $\boldsymbol{z_w} = 0$, we have the following relationship:

$$\begin{bmatrix} ku \\ kv \\ k \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.9}$$

being k a scale factor, $\boldsymbol{p^w} = [x, y, z]^T$ a 3D point on the book cover, and $\boldsymbol{u^c} = [u, v]^T$ its projection on the image.

Taking, as we defined, the origin of the world coordinate system in the center of the book cover, and knowing that its dimensions are $17.3cm \times 24.6cm$, we will use the DLT algorithm to retrieve the elements of $\mathbf{H}$.

The DLT algorithm is a pose estimation algorithm that assumes a set of known corre-

spondences $\{\mathbf{p}_i, \mathbf{u}_i\}$. For each correspondence, we can write the expression in 3.10.

$$\begin{bmatrix} k \cdot u_i \\ k \cdot v_i \\ k \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \tag{3.10}$$

Since, as explained in Section 1.2, we are studying the correlation between two planes, the elements corresponding to the z coordinate are eliminated, resulting in the equation 3.9.

Working with equation 3.9 to get rid of k, we get the expression in 3.11, that can be written as the linear system shown in 3.12.

$$\begin{aligned} u_i(h_{31}x_i + h_{32}y_i + h_{33}) &= h_{11}x_i + h_{12}y_i + h_{13} \\ v_i(h_{31}x_i + h_{32}y_i + h_{33}) &= h_{21}x_i + h_{22}y_i + h_{23} \end{aligned} \tag{3.11}$$

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_iu_i & -y_iu_i & -u_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -x_iv_i & -y_iv_i & -v_i \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ \vdots \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.12}$$

Having n correspondences we can form a bigger system (3.13) that is in fact a linear system $\mathbf{Mx} = \mathbf{0}$, where $\mathbf{M}$ is a $2n \times 12$ matrix.

$$\underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -y_1u_1 & -u_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1v_1 & -y_1v_1 & -v_1 \\ \vdots & & & & \vdots & & & & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_nu_n & -y_nu_n & -u_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -x_nv_n & -y_nv_n & -v_n \end{bmatrix}}_{\boldsymbol{M}} \underbrace{\begin{bmatrix} h_{11} \\ h_{12} \\ \vdots \\ h_{33} \end{bmatrix}}_{\boldsymbol{x}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.13}$$

The nontrivial solution to this linear system is the eigenvector associated to the smallest eigenvalue of $\mathbf{M}$. Since it is a nonsquare matrix, we use the *svd* function from Matlab (singular value decomposition).

In our case, the n correspondent points will be the four corners of the book. We know their 3D coordinates in centimeters in the world coordinate frame (deducted from the book size and the fact that the coordinate frame is located in its center, O), and we know their image coordinates in pixels, obtained directly by clicking on the correspondent image. Fig. 3.9 illustrates these correspondences for a given image. The real coordinates of the corners are always:

1. $(x_1, y_1) = (-8.65, 12.3)$ cm

2. $(x_2, y_2) = (8.65, 12.3)$ cm
3. $(x_3, y_3) = (8.65, -12.3)$ cm
4. $(x_4, y_4) = (-8.65, -12.3)$ cm

For the case of the figure, camera coordinates of the book corners are:

1. $(u_1, v_1) = (82.9255, 17.2234)$ pixels
2. $(u_2, v_2) = (115.3511, 49.3936)$ pixels
3. $(u_3, v_3) = (69.6489, 94.8404)$ pixels
4. $(u_4, v_4) = (37.4787, 62.9255)$ pixels

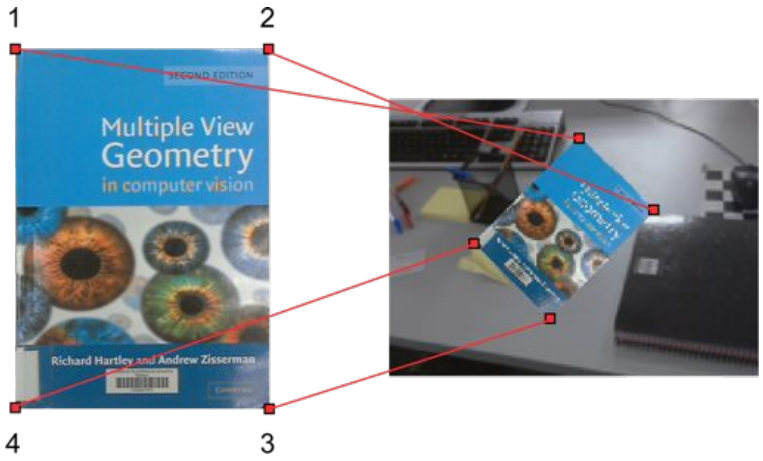

Figure 3.9: Correspondence between the book corners of the model in the *world* frame (left) and on the image plane or *camera* frame (right).

This is a tedious process that implies four clicks (one on each corner) per image. Nonetheless, thanks to the domain adaptation method, we will only need to do this with a few hundred images instead of thousands, taking advantage of the pre-trained network with synthetic images.

3.3 Label definition

When training a neural network, a correctly labeled dataset is the essential pre-requisite. From previous sections we obtained two image datasets (synthetic and real). In both cases, each image has an associated stored information: the pose of the camera (*virtual* or real) that took the picture with respect to the *world* coordinate frame.

Using that information to label the images is not as trivial as it might seem, since camera positions and rotations consist of 6 independent variables that take arbitrary values on a continuous domain. Hence the need of discretizing the domain is a natural choice.

The discretization adopted consists of dividing the the three-dimensional cartesian space in eight regular quadrants by dividing each axis in two regions. Axis $\boldsymbol{z_w}$ was divided in the value of 60, and axes $\boldsymbol{x_w}$ and $\boldsymbol{y_w}$ were both divided in 0. (see Fig. 3.10 for details). We represent them as cubes for simplicity, but all quadrants expand infinitely towards inf or $-$ inf of the corresponding axes.

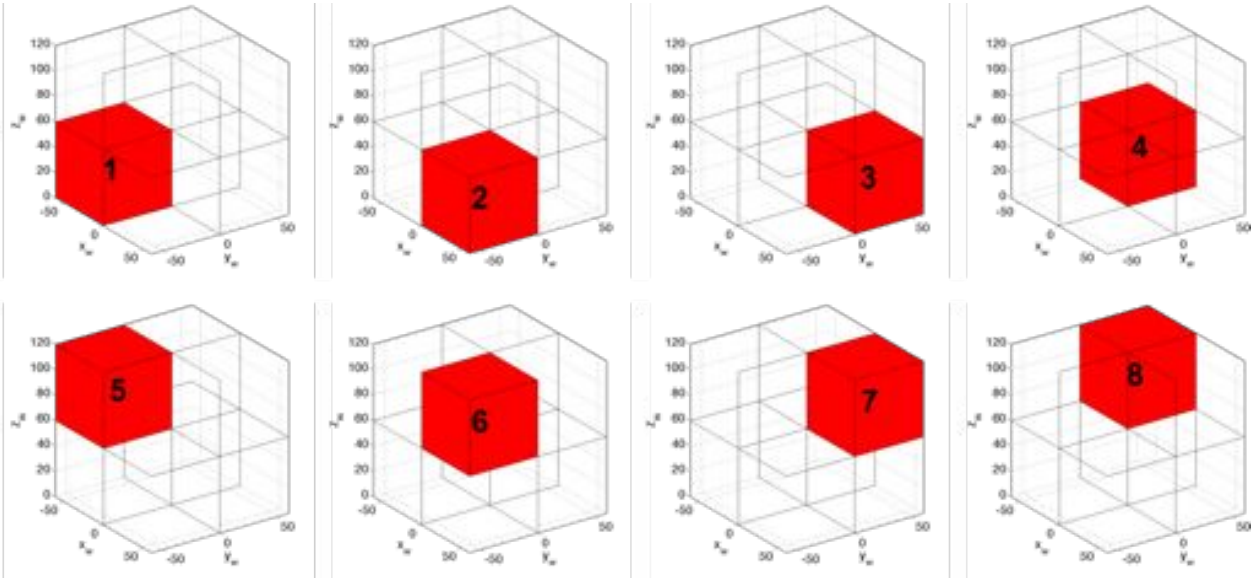

Figure 3.10: Quadrants in which we divide the space for labeling. First image shows a 3D view, and then two 2D views are added for clarity (in 2D views, grey numbers indicate 'hidden' quadrants).

Therefore, we will label each image according to the quadrant where the position of the correspondent camera ($\boldsymbol{c}$) lies. The fact that we ignore the rotation of the camera in this discretization comes from the definition of the center of our *world* coordinate frame, O. Since O is coincident with the center of the object (we defined it like that), and we estimate the camera pose with respect to the object, we assume that the camera is rotated always looking at the object, so positioning it in one of the eight defined quadrants implies a certain approximated rotation. If the camera were rotated pointing towards a point in a different direction, the object would not appear in the image and we would not be able to estimate the pose of the camera.

Apart from the eight labels for each one of the quadrants, there is another label (0) for images where the object is not present, i.e. the camera pose cannot be estimated. Fig. 3.11 shows some examples of images with labels 1-8. Note two phenomena:

- **Similarity between adjacent classes:** images near the classes frontiers can be very similar, since one image taken from a position $\boldsymbol{c} = (x, y, z)$ and another one from $\boldsymbol{c'} = (x', y', z')$, with $|x - x'| = \epsilon$, $|y - y'| = \epsilon$ or $|z - z'| = \epsilon$ (for small values of ϵ near the frontiers between classes) can be practically identical while having different labels (compare the last synthetic image of each row with the first one of the next one). For images where the target point is not exactly the center of the book this effect can be even more exaggerated.

- **Scale:** images of classes 1 and 5, 2 and 6, 3 and 7, 4 and 8 share ranges for x_w and y_w coordinates, being the only difference the range of the z_w coordinate. Therefore, one can think of the objects appearing on images labeled 5 to 8 as reduced versions of the objects apperaing on images labeled 1 to 4. This is clearly appreciable in Fig. 3.11, where top four rows show bigger representations of the book.

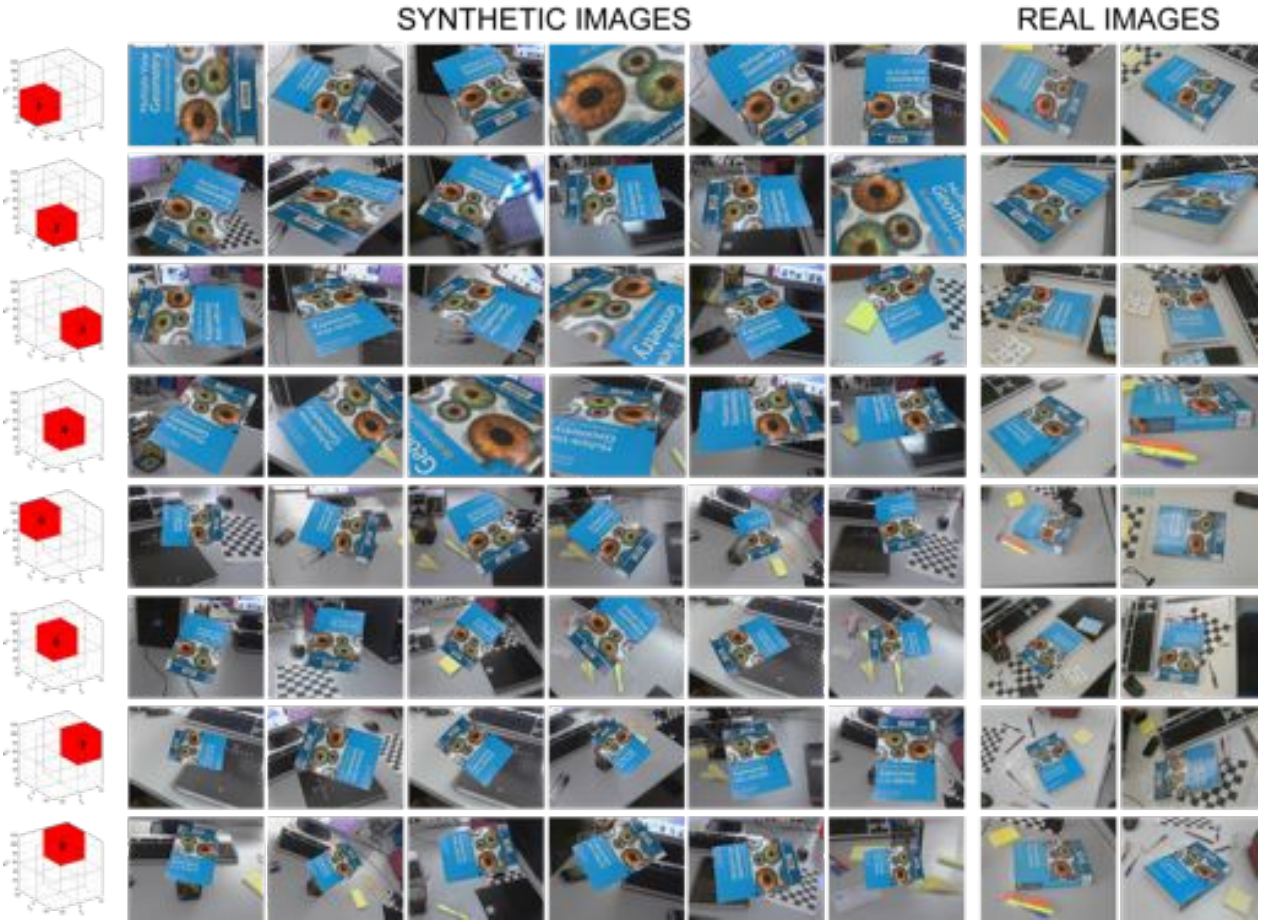

Figure 3.11: Examples of labeled images. All images in a row have the same label, indicated by the drawing on the left. Last two images of each row are real, while the rest are synthetic. Note the similarity between real and synthetic images with the same label.

Chapter 4

Convolutional Neural Network for Pose Estimation

Once the datasets have been detailed, it is time to focus on the machine learning part of the method that will use those datasets as inputs to both train and test the model.

4.1 Structure of the CNN

The Convolutional Neural Network (CNN) used in this work presents a typical structure for working with images, consisting on three layers, based on the network used in [31]. Since two image sizes are considered (160×120 and 80×60), two networks with identical structure but different size are created. Next, we describe the layers of the network. The filters sizes were chosen after testing with different values. We use here a generic image size $m \times n$ for the sake of generalization of the explanation. Fig. 4.1 shows a visual representation of the different layers affecting an input image.

1. The first layer applies 5×5 filters to the input $n \times m \times 3$ image, resulting in a $(n-4) \times (m-4) \times 64$ matrix, and then a L2-pooling function (with filter size equal to 4) to each one of the 64 results, giving as a result a $i \times j \times 64$ matrix, where $i = \frac{n-4}{4}$ and $j = \frac{m-4}{4}$.
2. Then, a second layer similar to the first one is applied again, obtaining a $\frac{(i-4)}{4} \times \frac{(j-4)}{4} \times 64$ matrix.
3. Later, that matrix is flattened to a one-dimensional vector of $\frac{\frac{(n-4)}{4}-4}{4} \times \frac{\frac{(m-4)}{4}-4}{4} \times 64$ components.
4. Finally, that vector is linearly reduced in two steps to the desired 9-component output vector.

This means that, depending on the input image size, we trained two different networks, being the sizes of the layers:

- $80 \times 60 \times 3$ (input image) $\rightarrow 76 \times 56 \times 64 \rightarrow 19 \times 14 \times 64 \rightarrow 15 \times 10 \times 64 \rightarrow 3 \times 2 \times 64 \rightarrow 384 \rightarrow 128 \rightarrow 9$
- $160 \times 120 \times 3$ (input image) $\rightarrow 156 \times 116 \times 64 \rightarrow 39 \times 29 \times 64 \rightarrow 35 \times 25 \times 64 \rightarrow 8 \times 6 \times 64 \rightarrow 3072 \rightarrow 1024 \rightarrow 9$

An example for a 80×60 input image is given in Fig. 4.1, with some samples of the intermediate layers of the network (9 of the 64 results are shown for each layer).

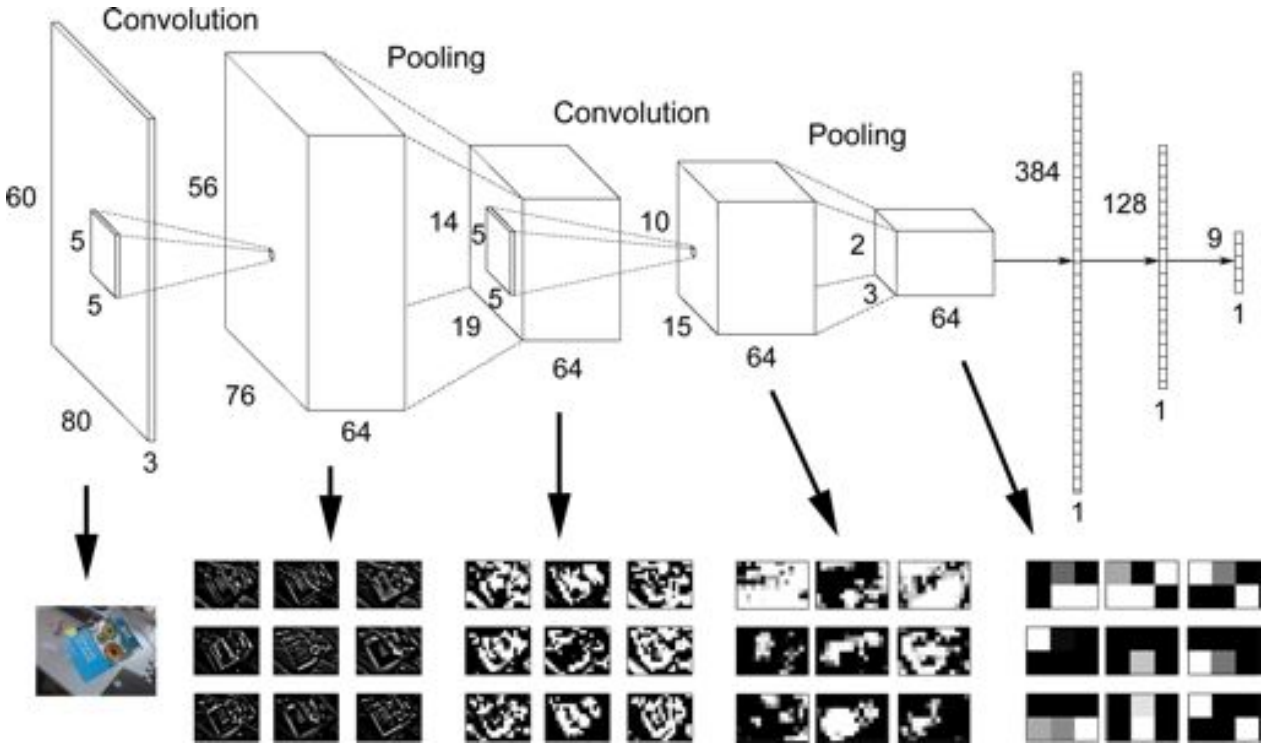

Figure 4.1: Scheme of the CNN used in the method. Numeric values are for a 80×60 input image. Some intermediate results of the network (9 out of 64 for each layer) are shown for several layers.

This type of network favors a large number of features over the density of connections. It locally normalizes the internal features at each stage, and smooths the result with the pooling functions.

4.2 Training process

Having defined the model, the next step is to define a loss function that will be minimized. Possibly the simplest one is the mean-square error between the outputs of the model (pre-

dictions $\boldsymbol{y}^n$) and the ground truth labels ($\boldsymbol{t}^n$), shown in Eq. 4.1

$$l\left(\boldsymbol{y}^n, \boldsymbol{t}^n\right) = \frac{1}{2}\sum_i \left(\boldsymbol{y}_i^n - \boldsymbol{t}_i^n\right)^2 \tag{4.1}$$

In our case, we will use the negative log-likehood function, because the mean-square error forces the model to predict the exact values imposed by the labels. In order to do that, we must turn the output of our model into normalized log-probabilities feeding them into a *softmax* function, which turns the linear regression into a logistic regression, as seen in Eq. 4.2, where $\boldsymbol{W}$ is a weight matrix and $\boldsymbol{b}$ a bias vector.

$$P\left(Y = i|\boldsymbol{x}^n, \boldsymbol{W}, \boldsymbol{b}\right) = softmax\left(\boldsymbol{W}\boldsymbol{x}^n + \boldsymbol{b}e\right) = \frac{e^{\boldsymbol{W}\boldsymbol{x}_i^n + \boldsymbol{b}}}{\sum_j e^{\boldsymbol{W}\boldsymbol{x}_j^n + \boldsymbol{b}}} \tag{4.2}$$

Then, the final prediction of our classification problem is obtained by taking the argument that maximizes that distribution (Eq. 4.3).

$$\boldsymbol{y}^n = arg\max_i P\left(Y = i|\boldsymbol{x}^n, \boldsymbol{W}, \boldsymbol{b}\right) \tag{4.3}$$

We want to maximize the likelihood of the correct class for each sample, i.e. minimize the negative log-likelihood or the cross-entropy between the predictions and the labels of the training data. Mathematically, the per-sample loss can be defined as stated in Eq. 4.4

$$l\left(\boldsymbol{x}^n, \boldsymbol{t}^n\right) = -\log\left(P\left(Y = \boldsymbol{t}^n|\boldsymbol{x}^n, \boldsymbol{W}, \boldsymbol{b}\right)\right) \tag{4.4}$$

Having the training data, the network to train and the loss function to minimize, we can start training.

It is important to notice that the optimization problem in supervised training of non-linear models is not convex, hence the need for a stochastic estimation of gradients, which produce better generalization results for several problems.

We will use Stochastic Gradient Descent (SGD) [27] given the non-convex nature of the problem. During the first epochs of the training (random initialization) no assumption should be made about the shape of the function, so often SGD is the best possible option. In addition, stochasticity is of capital importance in large convex problems, since it results in a much faster rough convergence.

4.3 Output of the CNN

When the network is already trained, it is ready to receive input images and obtain the corresponding outputs. In our network, the output is a vector of 9 components, each one

of them indicating the probability of the image belonging to one of the nine pose classes defined in Section 3.3. The position of the maximum of that vector directly indicates the label assigned to the input image (i.e. the quadrant where the camera is estimated to be).

4.4 Implementation details

These networks were implemented using Torch7 [2] as machine learning environment, and Lua [1] as scripting language. This language, quoting its webpage, *combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics.*

The training process of the networks was developed in a server with the following specifications:

- 2x Intel Xeon E5-2620V3 processors
- 64GB RAM - DDR4-2133 (8x 8GB)
- 4x 1TB HDD - SATA 6G @ 7.2Krpm 64M

It took around 26 days to perform all the training and testing processes in that server, taking into account the following figures:

- We trained 4 networks with 3000 synthetic images and then retrained them with 230 real images for domain adaptation.
- We performed 30 epochs of training for synthetic images, and 15 for domain adaptation for each network.
- Each training step is followed by a testing one. For testing we used 1000 synthetic images and 115 real images.
- We did all that for two image sizes: 80×60 (taking $\sim$ 15ms per image for training and ~ 7 for testing) and 160×120 (taking $\sim$ 71ms per image for training and $\sim$ 35ms for testing).

An approximated computational time for all this process can be obtained by the following sum of products:

$$\begin{aligned} 4 \text{ networks} \cdot [30 \text{ epochs} \cdot (3000 \text{ images} \cdot (71 \text{ ms} + 15 \text{ ms }) + 1000 \text{ images} \cdot (35 \text{ ms} + 7 \text{ ms})) + \\ + 15 \text{ epochs} \cdot (230 \text{ images} \cdot (71 \text{ ms} + 15 \text{ ms}) + 115 \text{ images} \cdot (35 \text{ ms} + 7 \text{ ms}))] = \\ = \mathbf{624.61 \text{ h} \simeq 26 \text{ days}} \end{aligned} \tag{4.5}$$

In Section 5.2 we explain the specifications of these networks.

4.5 Method variations: windowing approaches

With the tools previously presented, a camera pose can be correctly estimated. Nonetheless, we tried to optimize the result of our pose estimation problem via two object detection algorithms (sliding window and objectness) to see if there was an improvement in the results. This section describes how these approaches are included in the method.

Training images, as specified in Section 3.1.1, are only created from positions of the camera located in a limited portion of the space. Although this portion is selected according to the positions of the future real inputs of the network, and it might seem big enough, we contemplate the possibility of having real images where the camera is located further. In these images, the object will appear smaller on the image plane, with more background around it that in the training images, and the classification process might fail. In order to overcome this possible problem, we added a previous step to select uniquely the region with the highest probability of having the object, and use only this region as an input for the network.

Also, since the network is structured to receive an input image of a fixed size $m \times n$, in case the input image is bigger than that specified size, with this previous step we will only select as an input a region sized $m \times n$ instead of resizing the input image in order to be able to use the pre-trained networks.

In the following sections (4.5.1 and 4.5.2) we describe these approaches, to finally show a comparison of the visual results of both of them in 4.5.3.

Quantitative results are detailed in Chapter 5.

4.5.1 Sliding window

As we did in Section 4.1, since we work with two different image sizes (160×120 and 80×60), we use the generic size $m \times n$ in the explanation.

There exist several sliding window implementations, but the principle is always the same: to move a window of a fixed size smaller than the original image along all the image, normally also at different scales (see Fig. 4.2).

The parameters of the sliding window algorithm that we apply are (see Fig. 4.3):

- **Window size:** $w_x \times w_y = m \times n$, since we want it to act as input for the corresponding CNN.
- **Number of scales:** $N = 3\lfloor \log_2 m' - \log 2n' \rfloor$, where $m' \times n'$ is the size of the input image. Following [32], the size of the image in each scale i is determined by $[m'_i, n'_i] = 2^{i/3}\,[m', n']$, where $i = 0, 1, \ldots, N$

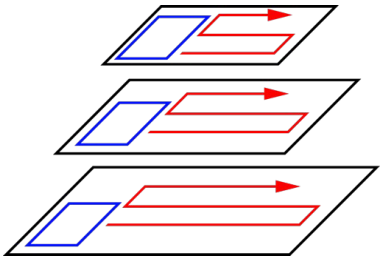

Figure 4.2: Basic sliding window procedure: a fixed size window (blue) moves along all the possible positions and scales of the image.

- **Step:** $s = 20px$. Instead of moving the window pixel by pixel, we use a step of 20 pixels to avoid a high computational cost.

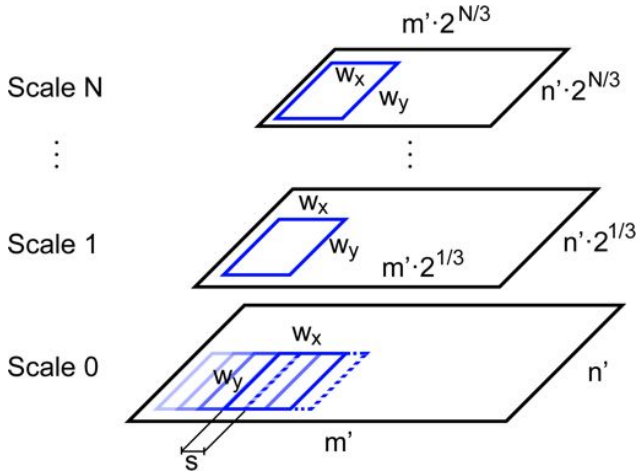

Figure 4.3: Visual description of the sliding window parameters.

For each scale, we evaluate every $m \times n$ window in our CNN, obtaining an estimated pose and a score (the maximum probability of the 9-component vector obtained from the network, as stated in Section 4.3), and ignoring it when the CNN determines that the object does not appear in the window (class 0). With that information, we construct a probability map and select the windows associated with the highest scores via a non-maxima supression algorithm. Finally, the label of the image is that of the window with the highest score remaining after the non-maxima suppression.

4.5.2 Objectness

Objectness measure [3] is a *class-generic* object detector that quantifies how likely it is for an image window to contain an object of any class thanks to a specific training focused on distinguish objects with a well-defined boundary from amorphous background elements. The measure combines characteristics of objects (such as appearing different from their surroundings and having a closed boundary) in a Bayesian framework.

Applying this algorithm to an input image returns a set of windows where it considers that the object appears. We then get these windows and evaluate them with our CNN, obtaining a probability map similar to the one obtained in previous section. We apply, as before, a non-maxima suppression algorithm to the map and get the window with the highest score.

The procedure here is slightly different than the one followed with sliding window. The main difference is that while in 4.5.1 all the windows from sliding window were evaluated by our CNN to obtain the probability map, now only the windows obtained via the objectness algorithm are, so the number of calls to Torch is significantly smaller.

4.5.3 Sliding window and objectness comparison

A visual comparison of both detection algorithms (considering that our CNN acts as a detection algorithm for the object for sliding window, because it is able to detect whether the object is present or not in the image) is shown in Fig. 4.4.

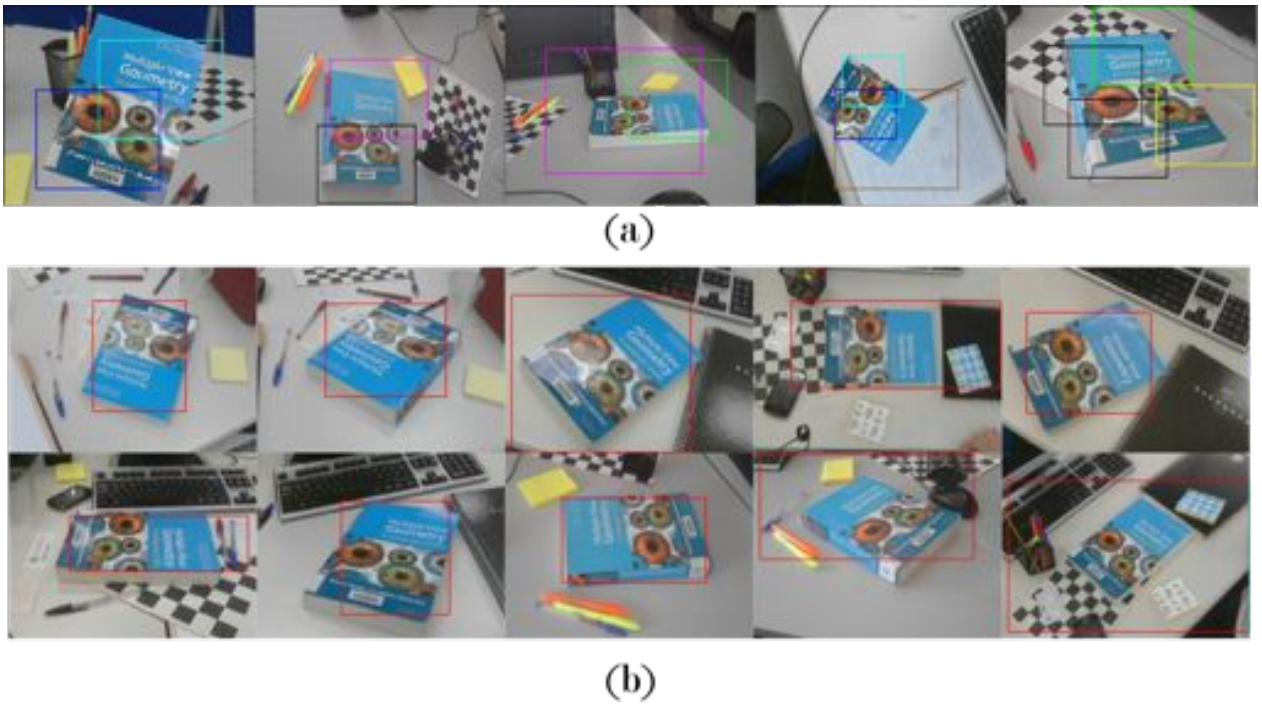

Figure 4.4: Boxes returned by sliding window (a) and objectness (b) after the non-maxima suppression step. Note how the objectness method generally detects the whole object, while the sliding window method tends to find parts of it. Different colors in the sliding window boxes indicate differently labeled results with high probability of belonging to one of the classes 1-8.

Chapter 5

Results

In this chapter we analyze with detail all the results obtained with the method tested in different datasets and using differently trained networks. Firstly, we describe the used datasets and the networks. Afterwards, the results are presented, and finally the method is compared against a feature-based pose estimation method.

5.1 Datasets

Twenty-four different datasets were created to test the method. At the end of this section we will give a short name for each one of them in order to simplify the results tables of this chapter.

1. **Image size:** In this work, we decided to use two different image sizes in order to see how this affects the performance of the method. To accelerate the training process and prove one of the points of the method (that it can work with low resolution images), small images were used in both cases.
   - **Size 1:** 80×60 pixels.
   - **Size 2:** 160×120 pixels.
2. **Synthetic and real:** As previously stated, we used thousands of synthetic images and hundreds of real images. Generation of synthetic images is explained in Section 3.1, and labeling of real ones in 3.2.
   - **Type 1:** Synthetic images.
   - **Type 2:** Real images.
3. **Target centered on image:** For the synthetic images, as mentioned before, we generated two different datasets with only one difference: the point the camera is looking

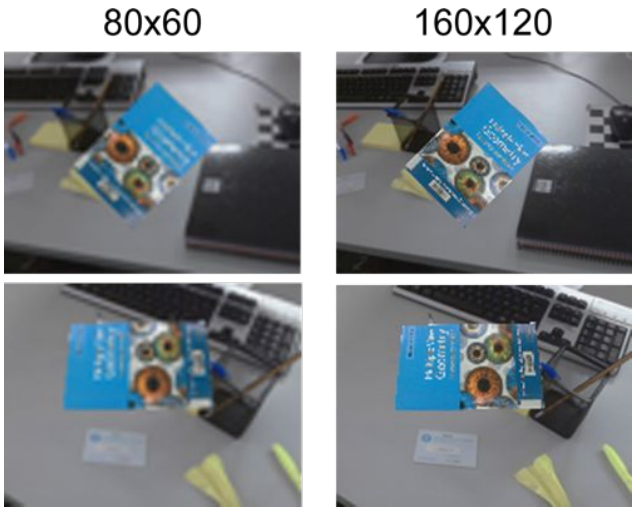

Figure 5.1: Example images of both sizes. We zoomed in the small images to ease the visual comparison of the resolutions.

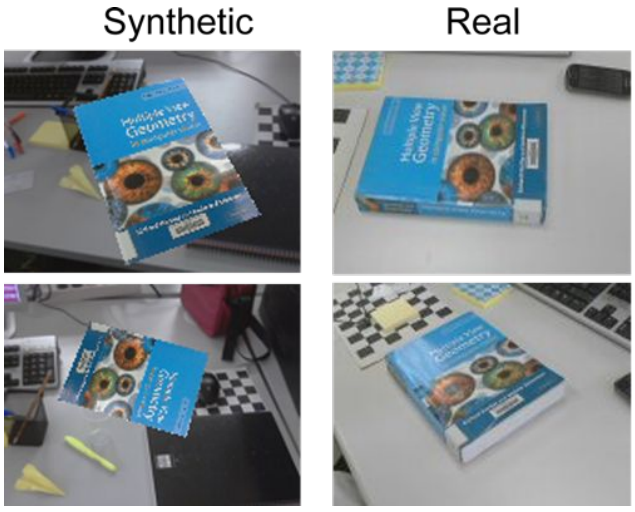

Figure 5.2: Example images of synthetic and real datasets.

at ($\boldsymbol{t}$). In one of the datasets, the camera points always exactly at the center of the book (that coincides with the center of the *world* coordinate frame). In the other one, it points at a random point in a 6×6 *cm* region around that point, so the book does not appear exactly centered in the resulting image. The aim of this modification is to create synthetic images similar to the future real input images to perform a more adequate training process.

- **Target 1:** center of *world* coordinate frame, O.
- **Target 2:** random point around O.

4. **Blur:** Combining the three previous modifications, we obtain six datasets. To check the robustness of the method against bad quality images, we added three different levels of motion blur to every image of each dataset, creating 6×3 new datasets. In order

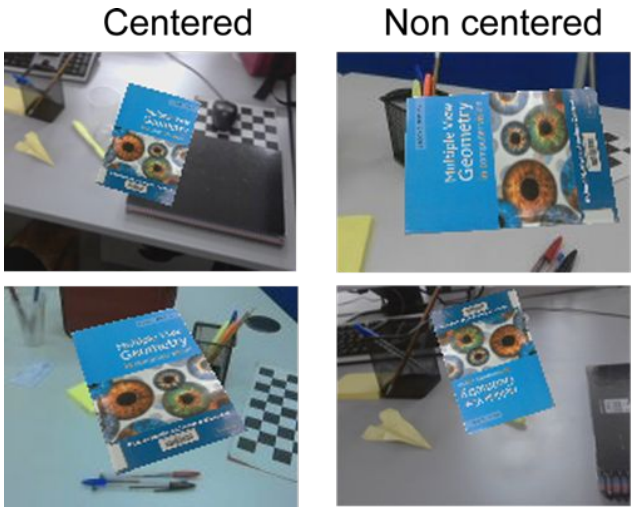

Figure 5.3: Example images of centered and non-centered datasets.

to do that, we simply apply a rotationally symmetric Gaussian lowpass filter to the images. Size and σ of the filters is chosen as follows:

- **Blur level 1:** 2×2, $\sigma = 1$ for **Size 1** images / 2×2, $\sigma = 2$ for **Size 2** images.
- **Blur level 2:** 4×4, $\sigma = 2$ for **Size 1** images / 5×5, $\sigma = 3$ for **Size 2** images.
- **Blur level 3:** 6×6, $\sigma = 3$ for **Size 1** images / 10×10, $\sigma = 5$ for **Size 2** images.

Visual results of applying this filter are shown in Fig. 5.4.

Figure 5.4: From left to right: original image, image with **Blur level 1**, image with **Blur level 2** and image with **Blur level 3**.

5. **Nomenclature:** Nomenclature of every method described in this section for using in further figures is shown in Table 5.1. Taking into account that every synthetic dataset consists of 5000 images, and every real dataset has 460 images, we decided to use 60% of the samples for training, 20% for testing and 20% for validating.

Dataset name	Synthetic/ Real	Image size	Centered target	Blur level	Train. imgs.	Test imgs.	Val. imgs
C_1	S	1	Yes	0	3000	1000	1000
C_1b1	S	1	Yes	1	3000	1000	1000
C_1b2	S	1	Yes	2	3000	1000	1000
C_1b3	S	1	Yes	3	3000	1000	1000
noC_1	S	1	No	0	3000	1000	1000
noC_1b1	S	1	No	1	3000	1000	1000
noC_1b2	S	1	No	2	3000	1000	1000
noC_1b3	S	1	No	3	3000	1000	1000
C_2	S	2	Yes	0	3000	1000	1000
C_2b1	S	2	Yes	1	3000	1000	1000
C_2b2	S	2	Yes	2	3000	1000	1000
C_2b3	S	2	Yes	3	3000	1000	1000
noC_2	S	2	No	0	3000	1000	1000
noC_2b1	S	2	No	1	3000	1000	1000
noC_2b2	S	2	No	2	3000	1000	1000
noC_2b3	S	2	No	3	3000	1000	1000
r_1	R	1	-	0	230	115	115
r_1b1	R	1	-	1	230	115	115
r_1b2	R	1	-	2	230	115	115
r_1b3	R	1	-	3	230	115	115
r_2	R	2	-	0	230	115	115
r_2b1	R	2	-	1	230	115	115
r_2b2	R	2	-	2	230	115	115
r_2b3	R	2	-	3	230	115	115

Table 5.1: Dataset names and characteristics. Note that for real images there is no "Centered target" parameter, since we do not choose if the camera points exactly at the center of the *world* origin or not. Last columns show number of images used for training, testing and validating the networks.

5.2 Networks

Now that all the datasets used to test the method are defined, we describe the differently trained networks.

- **Network A:** trained with 3000 images from noC_1 dataset. Tested with 1000 images.
- **Network B:** trained with 3000 images from C_1 dataset. Tested with 1000 images.
- **Network C:** trained with 3000 images from noC_2 dataset. Tested with 1000 images.
- **Network D:** trained with 3000 images from C_2 dataset. Tested with 1000 images.

5.3 Domain adaptation

Domain adaptation is a machine learning technique that focuses on using the stored knowledge gained while solving one problem to solve a different but related problem. In our case, we want our networks (trained with synthetic images) to work properly with real input images, so we fine-tune them with a small set of real images, i.e. we run the training process with backpropagation to slightly modify the weights of the neurons of the network.

The four networks previously trained were later retrained with 230 real images (60% of the dataset) of the right size, obtaining:

- **Network Ar:** network A retrained with 230 images from r_1 dataset. Tested with 115 images.
- **Network Br:** network B retrained with 230 images from r_1 dataset. Tested with 115 images.
- **Network Cr:** network C retrained with 230 images from r_2 dataset. Tested with 115 images.
- **Network Dr:** network D retrained with 230 images from r_2 dataset. Tested with 115 images.

5.4 Result analysis

In this section we compare the results of evaluating every dataset with every network (when necessary, input images are resized to match the input size required by the network). We also compare with the results using sliding window and objectness. The comparison is based in several metrics: the percentage of success of the predictions, the precision, the recall, the f-measure, the confusion matrix and the time per image evaluation. We also compute this metrics for the datasets with different blur levels.

All these results are evaluated over a validation set of images of each dataset, consisting on 1000 images for the synthetic datasets and 115 images for real datasets.

Note how synthetic datasets obtain generally better values than the real ones, and how domain adaptation improves those values for real images while decreases the values for synthetic.

5.5 Overall accuracy

We show in Fig. 5.5 the percentage of correctly predicted labels for each dataset for two trained networks, with and without using sliding window and objectness, and for different levels of blurring. Complete results for all datasets and networks can be found in A.1.

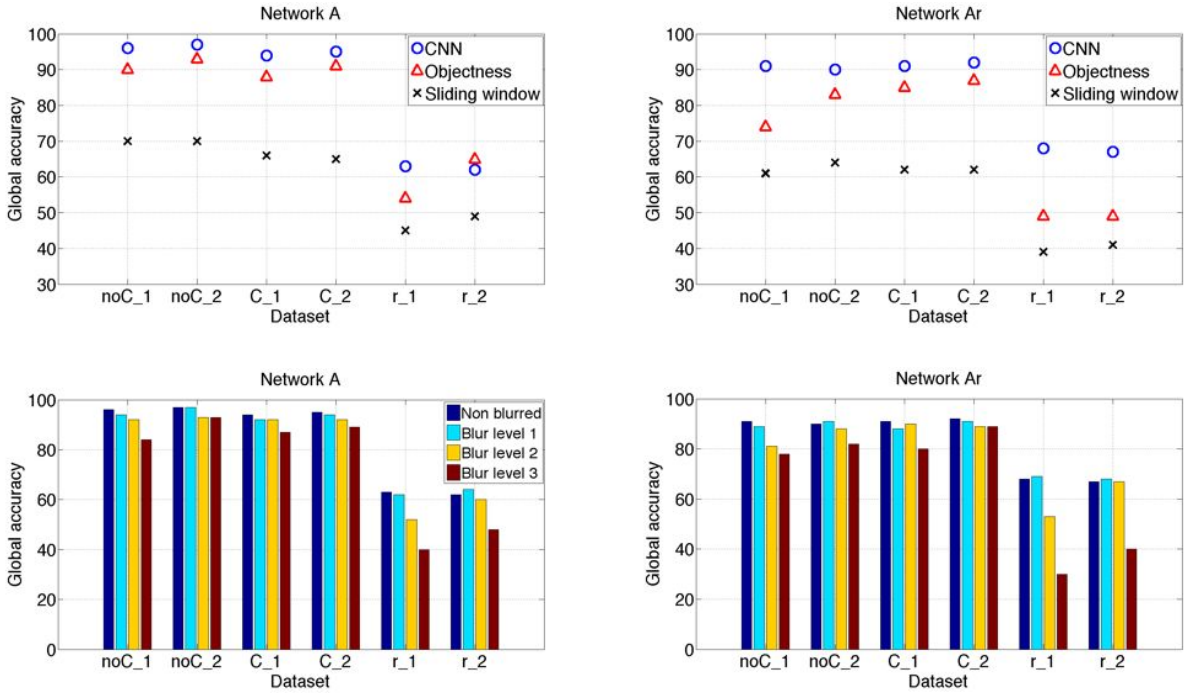

Figure 5.5: Global accuracy (%) for all datasets for networks A and Ar. Top: proposed method ('CNN'), and proposed method combined with sliding window and objectness, as stated in Chapter 4.5. Bottom: for the proposed method ('CNN'), global accuracy for each dataset with different blur levels compared with the dataset without blurring.

A first look reveals the futility of applying sliding window or objectness techniques. The network itself produces almost always a better result than combined with any of these two methods.

Left-hand images correspond to a network trained only with synthetic images, while right-hand images correspond to the same network retrained with some real images (as explained in 5.3). We observe how the performance of the two real datasets (r_1 and r_2) improves more than 5% with the domain adaptation.

Note how the effect of blurring is not critical in most cases. Only the hardest level of blurring affects significantly the result in real datasets.

Accuracy is also evaluated for each one of the nine classes (negative image plus eight position labels). Fig. 5.6 shows accuracy for each class for the non-centered 160×120 dataset *noC_2* (also with different levels of blurring) evaluated with the proposed method (using only the neural network) and with the method combined with sliding window and objectness.

Complete results can be found in A.2.

Note how the accuracy is always higher than 95% for class 0 (image without the object), indicating that the network is able to detect whether the object is present in the picture or not in a quite robust way, which is indeed the *classification problem*. Again, we see that the performance with sliding window and objectness is worse than without them, and that until a certain level of blurring the accuracy of the method remains practically identical for all classes.

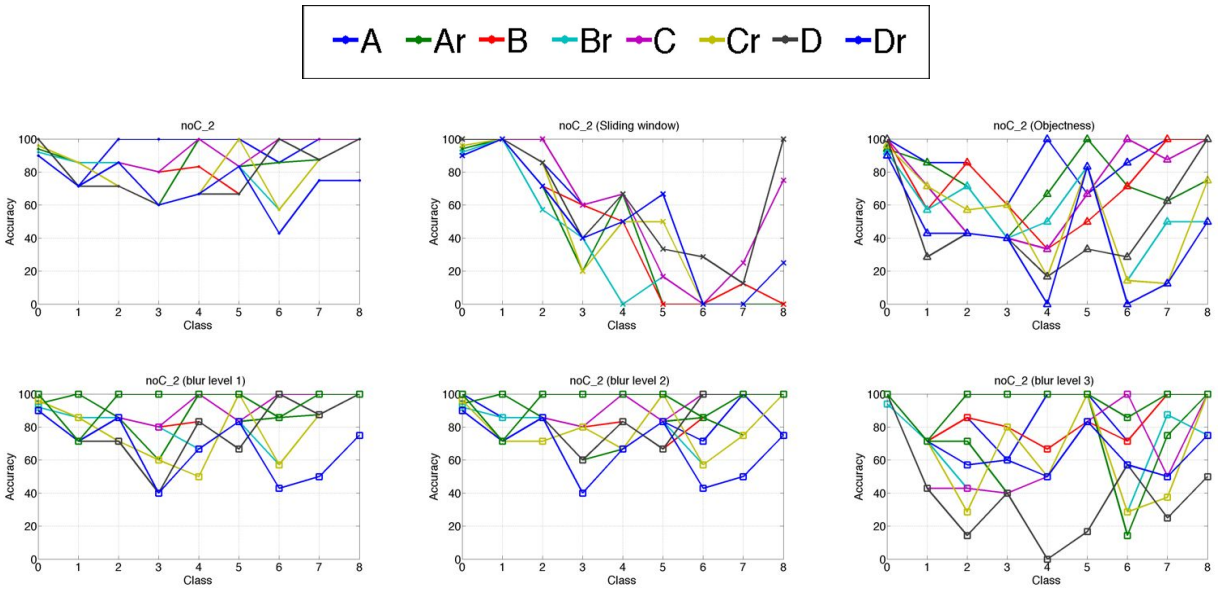

Figure 5.6: Accuracy (% of correct predictions) for each class (0-8). Each line represents the value computed for a different network.

5.6 Precision and recall: F-measure

Precision and recall are well-known metrics for classification problems. They are based on the number of false positives (FP), false negatives (FN) and true positives (TP) over several samples, and defined in Eq. 5.1. In order to compute these values, we treat each class as a binary classification problem, defining a positive sample when it belongs to that class, and negative when not (i.e. when it belongs to any of the other eight classes).

$$\begin{aligned} \text{precision} &= \frac{TP}{(TP+FP)} \\ \text{recall} &= \frac{TP}{(TP+FN)} \end{aligned} \tag{5.1}$$

In order to simplify the results, we combine precision and recall in a single measure that is the harmonic mean of both, the F-measure (see Eq. 5.2). Fig. 5.7 shows values of F-measure

for each class for the non-centered 80×60 dataset *noC_1*.

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{5.2}$$

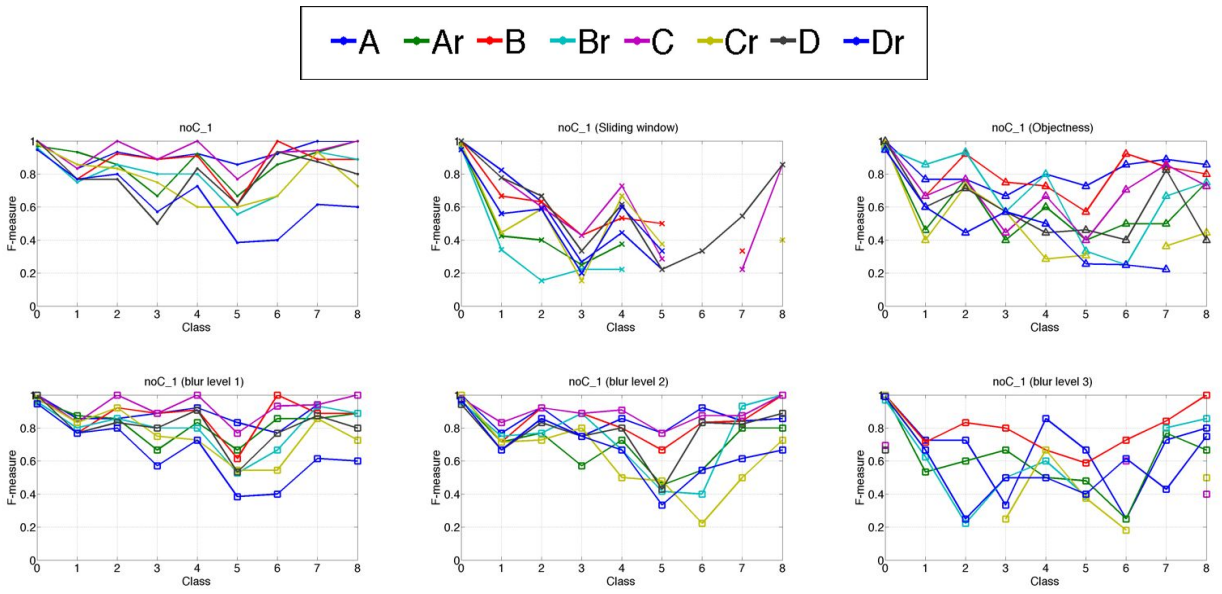

Figure 5.7: F-measure for each class (0-8). Each line represents the value computed for a different network.

5.7 Confusion matrix

With the visual representation of the confusion matrices, we can have a more general idea of the results at a glance. Rows of these matrices correspond to the ground truth label of the samples, and columns indicate the label assigned by the method. Therefore, each cell (i, j) represents the percentage of samples of class i that are classified as class j. Lighter colors indicate higher values (white represents a value of 1, black represents a value of 0).

Ideally, these matrices would be identity matrices (black matrices with white diagonals). The results obtained are not perfect identity matrices, but in most cases they follow the same pattern of diagonal elements higher than the rest of elements (see Appendix A.3 for complete results).

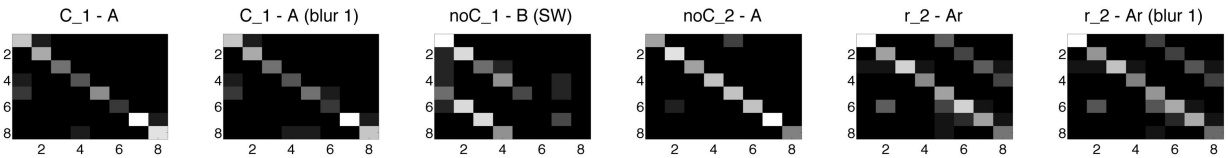

Figure 5.8: Confusion matrix for classes (1-8) for some datasets and networks.

Values for negative samples are not represented here. These values, as stated before, are always higher than 95% for class 0, so they provoked the rest of cells to be too dark to appreciate the differences between them. Fig. 5.8 shows a phenomenon that can explain some low values obtained in the global accuracy section. Note that in some cases the matrices present two white parallel lines. These lines indicate that a sample with ground truth label 1 is many times classified as class 5, and same happens for classes 2 and 6, 3 and 7 and 4 and 8. If we refer to Fig. 3.10, we can see how these classes share x_w and y_w coordinates, being the only difference the z_w value, so these mistakes can come from images that are visually very similar (the positions of the camera can be near $z_w = 60$, the limit we imposed between classes).

This means that even if the overall accuracy values are not very high in some cases because the predicted class does not match the ground truth one, the label given by the network is not meaningless, since it can be classifying an image in an adjacent quadrant that had some very similar training images.

5.8 Computational time per image

Computational time per image is generally smaller than 0.3 ms for the neural network, but it critically increases when applying a previous step like sliding window or objectness (taking into account that these steps are implemented in Matlab). So not only these steps decrease the accuracy, but also increase the computational time. See complete results in Appendix A.5.

Note also that the method is slightly faster for blurred images.

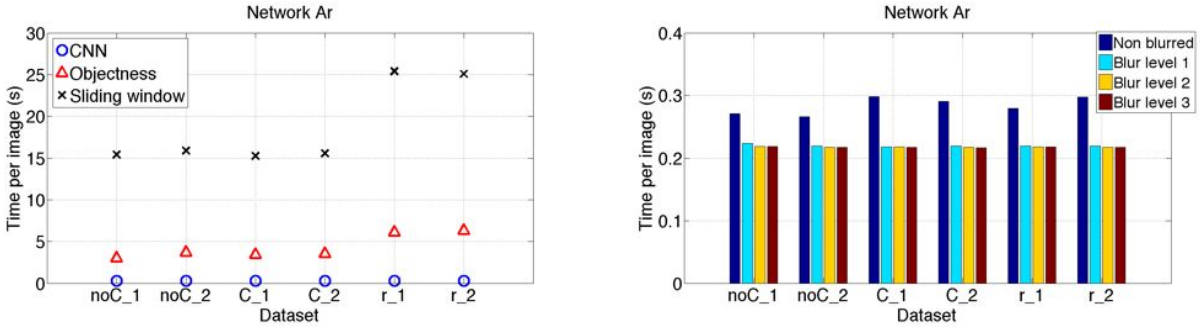

Figure 5.9: Computational time per image for different methods and blur levels.

5.9 Evaluation with a state-of-the-art method

The same metrics were computed evaluating all the datasets with a state-of-the-art feature-based method. We chose to use RANSAC combined with Hager's PnP [21]. Results are shown in Figs. 5.10, 5.11 and 5.12. This method is applied only to images labeld 1 to 8,

where the object appears. Results of applying this method to our dataset are not very good compared to the proposed method: the low-resolution images that we are using (and that make the CNN training process be faster) do not offer the quality required for a good feature extraction. This method is also one magnitude order slower than the proposed one.

Only the first level of blurring is shown in the results, since for bigger levels results are so poor that it does not make sense to represent them. Observe that confusion matrices for blurred datasets indicate that the method failed for all the samples.

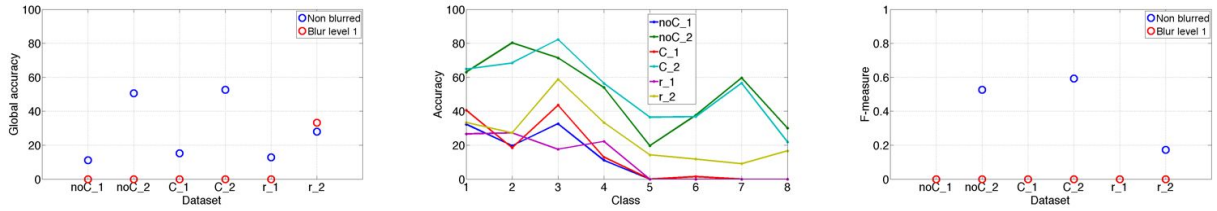

Figure 5.10: From left to right: global accuracy for all datasets using RANSAC + Hager's PnP, per-class accuracy for classes (1-8) for all datasets and F-measure for all datasets.

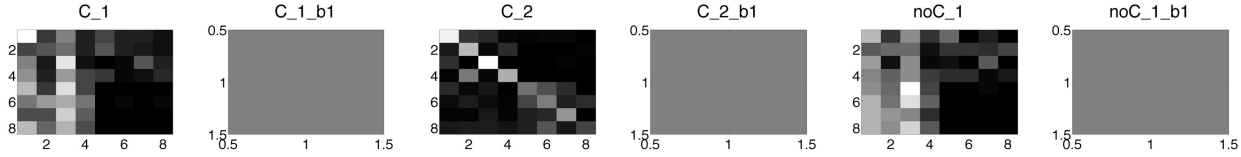

Figure 5.11: Confusion matrix for classes (1-8) for some datasets. All matrices are represented in A.6

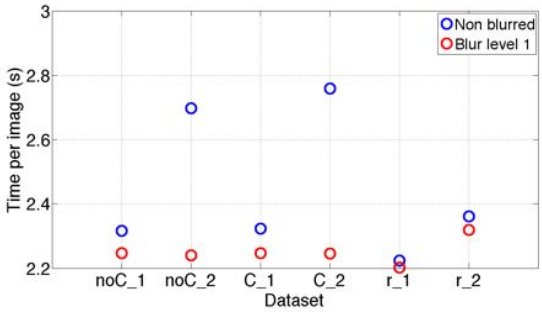

Figure 5.12: Computational times of applying RANSAC + Hager's PnP to each dataset.

Chapter 6

Conclusions and future work

6.1 Conclusions

We set up a framework for pose estimation using convolutional neural networks, and exhaustively tested it against different datasets, also combined with windowing methods and using blurred images. Some interesting conclusions can be extracted from all these experiments.

In the first place, the best results we obtained are, as expected, those of the synthetic images, since the networks are mostly trained with that kind of images. Nonetheless, for many cases the pose estimation of real images was correct, and we discussed what could be happening when not (the network was generally assigning labels from the adjacent classes).

We discovered that the proposed apperance-based method does not need a previous step to constrain the parts of the image where it looks for the object. Moreover, the method proved itself relatively robust to blurring in the input image.

In addition, when analyzing the results, we saw that the *binary classification problem* was correctly solved for over 95% of the samples (class 0).

In conclusion, we demonstrated that convolutional neural networks can be used to solve a discretized pose estimation problem in a more robust but less precise (due to discretization) way than feature-based methods, and that synthetically generated images produce acceptable results that improve when receiving some help coming from domain adaptation.

6.2 Future work lines

Once we demonstrated that the method works with high accuracy for synthetic input images and with a reasonable precision for real images, and that the domain adaptation improves

its behavior against real inputs, some interesting future steps would be:

- Testing the method with new objects, specially three-dimensional objects.
- Applying domain adaptation to the current networks trying to estimate the pose of a planar object with a different texture than the one we currently use (a book with a different cover, for instance).
- Dividing the pose space into smaller regions in order to increase the precision.
- Implementing a live version for running in real time (e.g. to track the camera pose in a video).

Bibliography

[1] Lua - The programming language. `http://www.lua.org/`.

[2] Torch - A scientific computing framework for LuaJIT. `http://torch.ch/`.

[3] Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. Measuring the objectness of image windows. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(11):2189–2202, 2012.

[4] Blender Online Community. Blender - a 3d modelling and rendering package. `http://www.blender.org`.

[5] D. Dai, H. Riemenschneider, and L. Van Gool. The synthesizability of texture examples. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[6] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in Neural Information Processing Systems*, pages 2366–2374, 2014.

[7] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1915–1929, 2013.

[8] Luis Ferraz, Xavier Binefa, and Francesc Moreno-Noguer. Very fast solution to the pnp problem with algebraic outlier rejection. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 501–508. IEEE, 2014.

[9] Philipp Fischer, Alexey Dosovitskiy, and Thomas Brox. Descriptor matching with convolutional neural networks: a comparison to sift. *arXiv preprint arXiv:1405.5769*, 2014.

[10] Yaroslav Ganin and Victor Lempitsky. N^ 4-fields: Neural network nearest neighbor fields for image transforms. In *Computer Vision–ACCV 2014*, pages 536–551. Springer, 2014.

[11] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jagannath Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE, 2014.

[12] Daniel Glasner, Meirav Galun, Sharon Alpert, Ronen Basri, and Gregory Shakhnarovich. Viewpoint-aware object detection and pose estimation. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 1275–1282. IEEE, 2011.

[13] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. *arXiv preprint arXiv:1411.5752*, 2014.

[14] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[15] Wenze Hu and Song-Chun Zhu. Learning a probabilistic model mixing 3d and 2d primitives for view invariant object recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2273–2280. IEEE, 2010.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[17] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[18] V. Lepetit, F. Moreno-Noguer, and P. Fua. Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision (IJCV)*, 81(2):155–166, 2009.

[19] Joerg Liebelt and Cordelia Schmid. Multi-view object class detection with a 3d geometric model. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1688–1695. IEEE, 2010.

[20] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *arXiv preprint arXiv:1411.4038*, 2014.

[21] Chien-Ping Lu, Gregory D Hager, and Eric Mjolsness. Fast and globally convergent pose estimation from video images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(6):610–622, 2000.

[22] Francesc Moreno-Noguer, Vincent Lepetit, and Pascal Fua. Accurate non-iterative o (n) solution to the pnp problem. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[23] Mustafa Ozuysal, Vincent Lepetit, and Pascal Fua. Pose estimation for category specific multiview object localization. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 778–785. IEEE, 2009.

[24] Nadia Payet and Sinisa Todorovic. From contours to 3d object detection and pose estimation. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 983–990. IEEE, 2011.

[25] Adrian Penate-Sanchez, Juan Andrade-Cetto, and Francesc Moreno-Noguer. Exhaustive linearization for robust camera pose and focal length estimation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(10):2387–2400, 2013.

[26] Adrian Penate-Sanchez, Francesc Moreno-Noguer, Juan Andrade-Cetto, and Francois Fleuret. Letha: Learning from high quality inputs for 3d pose estimation in low quality images. In *3D Vision (3DV), 2014 2nd International Conference on*, volume 1, pages 517–524. IEEE, 2014.

[27] Avi Pfeffer. CS181 Lecture 5 — Perceptrons. *Harvard University*.

[28] E. Simo-Serra, E. Trulls, L. Ferraz, I. Kokkinos, P. Fua, and F. Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2015.

[29] Hao Su, Min Sun, Li Fei-Fei, and Silvio Savarese. Learning a dense multi-view representation for detection, viewpoint classification and synthesis of object categories. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 213–220. IEEE, 2009.

[30] Alexander Thomas, Vittorio Ferrar, Bastian Leibe, Tinne Tuytelaars, Bernt Schiel, and Luc Van Gool. Towards multi-view object class detection. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1589–1596. IEEE, 2006.

[31] Torch Online Community. Machine learning with Torch7. Step 3: Loss Function. `http://code.madbits.com/wiki/doku.php?id=tutorial_supervised_3_loss`. Accesed: 2015-03-03.

[32] Michael Alejandro Villamizar Vergel. *Efficient Approaches for Object Class Detection*. PhD thesis, Universitat Politècnica de Catalunya, 2012.

[33] Vicon Motion Systems Ltd. Vicon system. `http://www.vicon.com/`.

[34] Michael Alejandro Villamizar Vergel, Helmut Grabner, Juan Andrade-Cetto, Alberto Sanfeliu Cortés, Luc Van Gool, Francesc Moreno-Noguer, et al. Efficient 3d object detection using multiple pose-specific classifiers. 2011.

[35] Paul Wohlhart and Vincent Lepetit. Learning descriptors for object recognition and 3d pose estimation. *arXiv preprint arXiv:1502.05908*, 2015.

[36] Jure Žbontar and Yann LeCun. Computing the stereo matching cost with a convolutional neural network. *arXiv preprint arXiv:1409.4326*, 2014.

Appendix A

Complete results

A.1 Global Accuracy

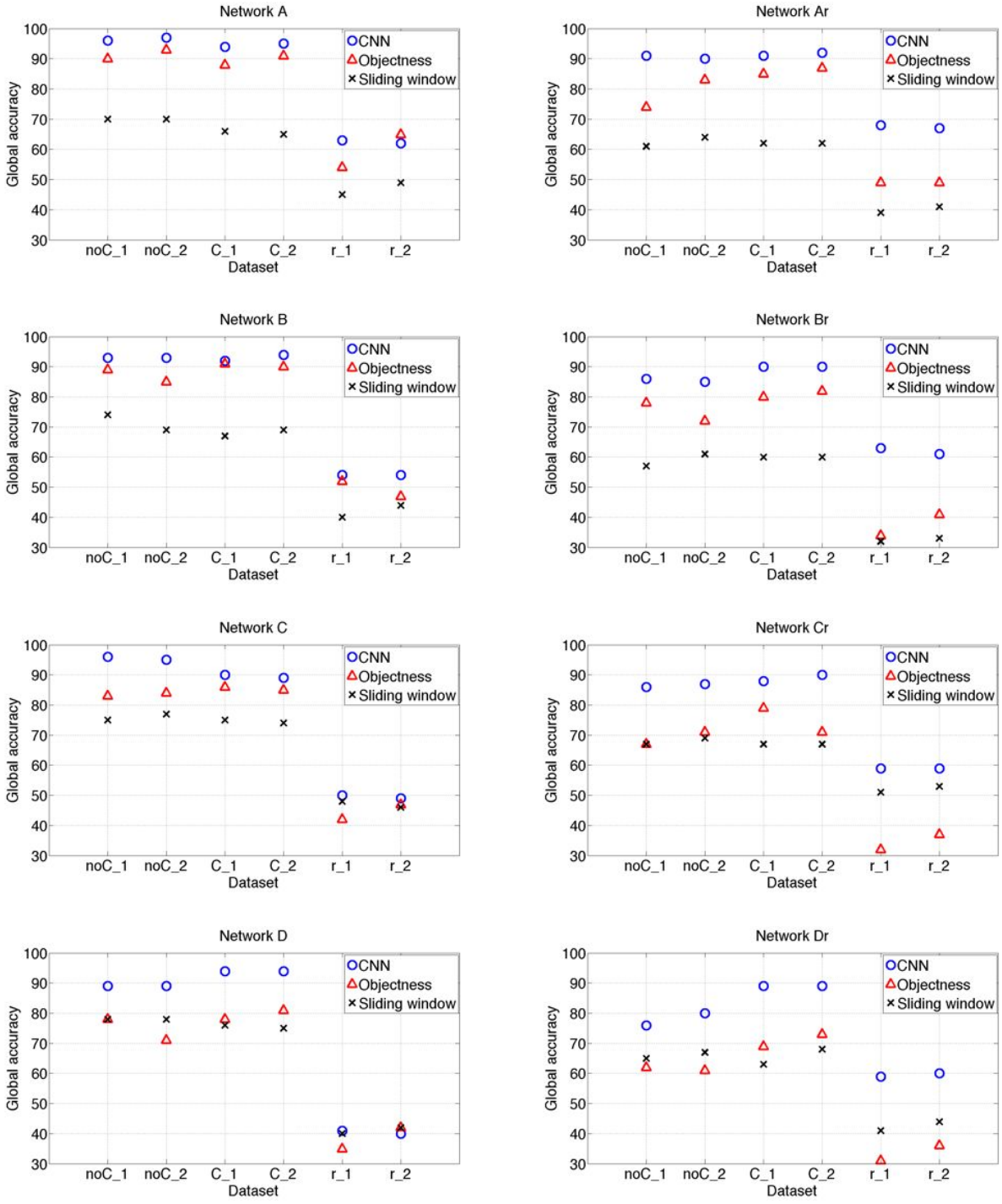

Figure A.1: Global accuracy (%) for all datasets and networks. Results for the proposed method ('CNN') and for the proposed method combined with sliding window and objectness, as stated in Chapter 4.5.

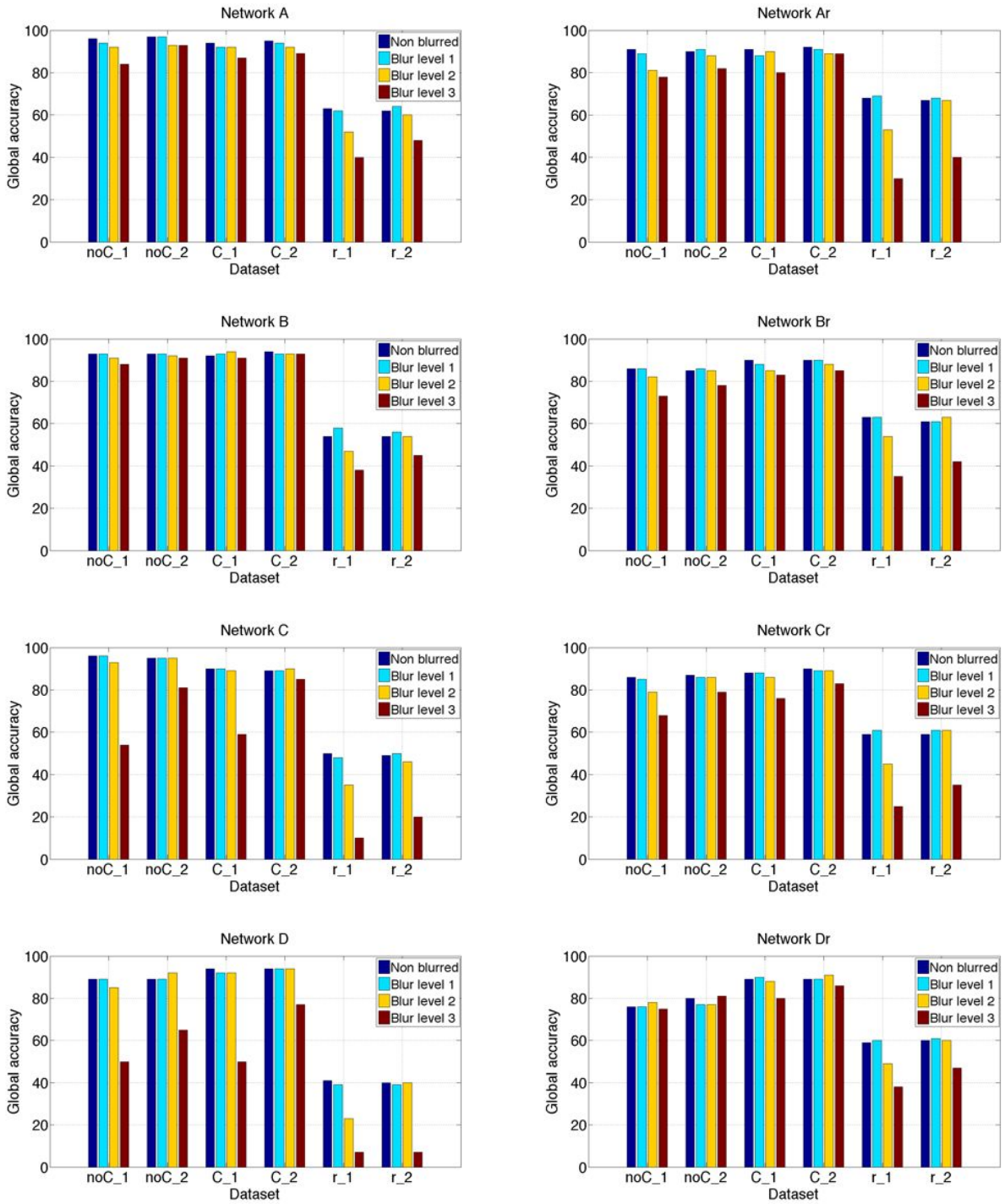

Figure A.2: For the proposed method ('CNN'), global accuracy for each dataset with different blur levels compared with the dataset without blurring.

A.2 Per-class Accuracy

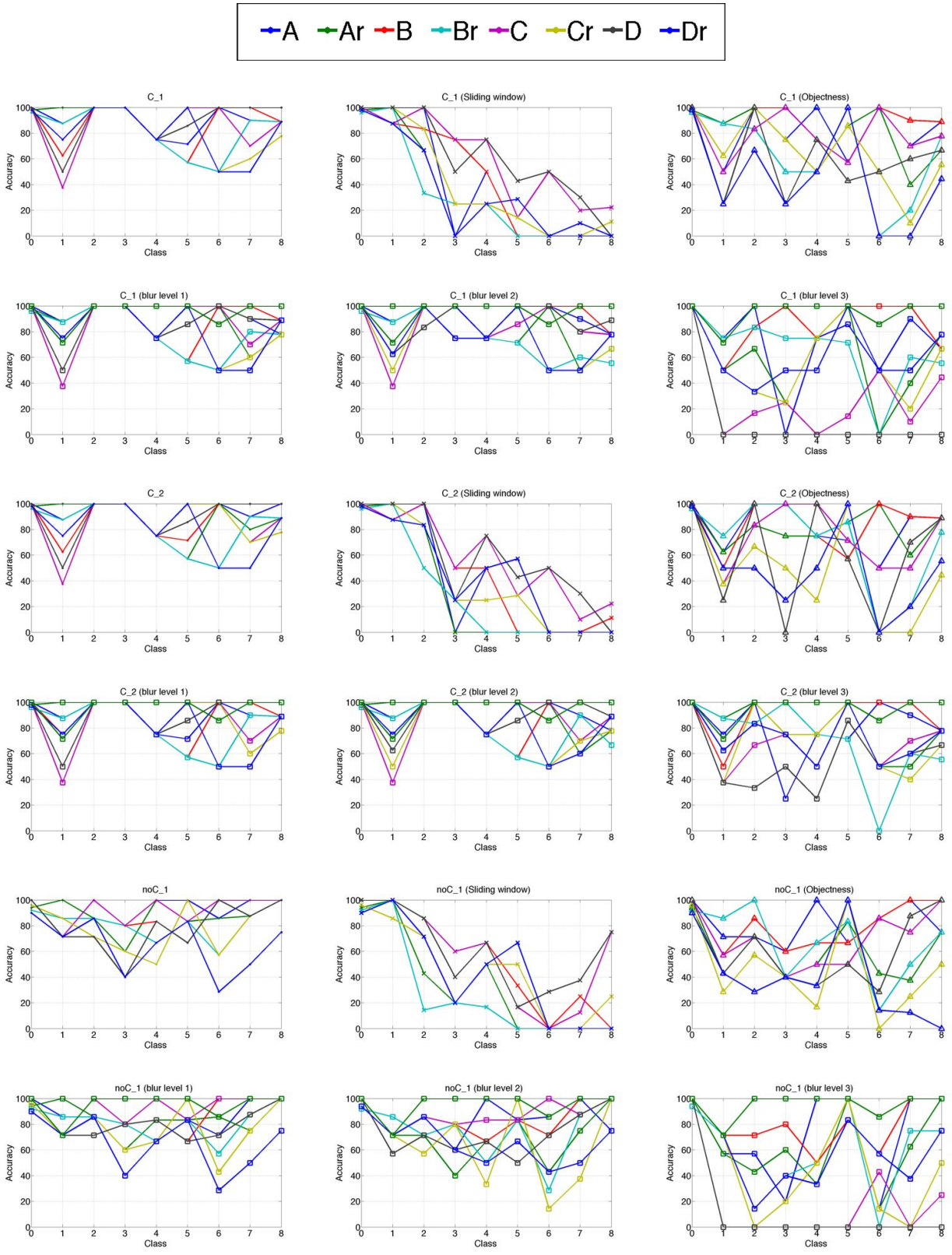

Figure A.3: For each dataset, accuracy (% of correct predictions) for each class (0-8). Each line represents the value computed for a different network.

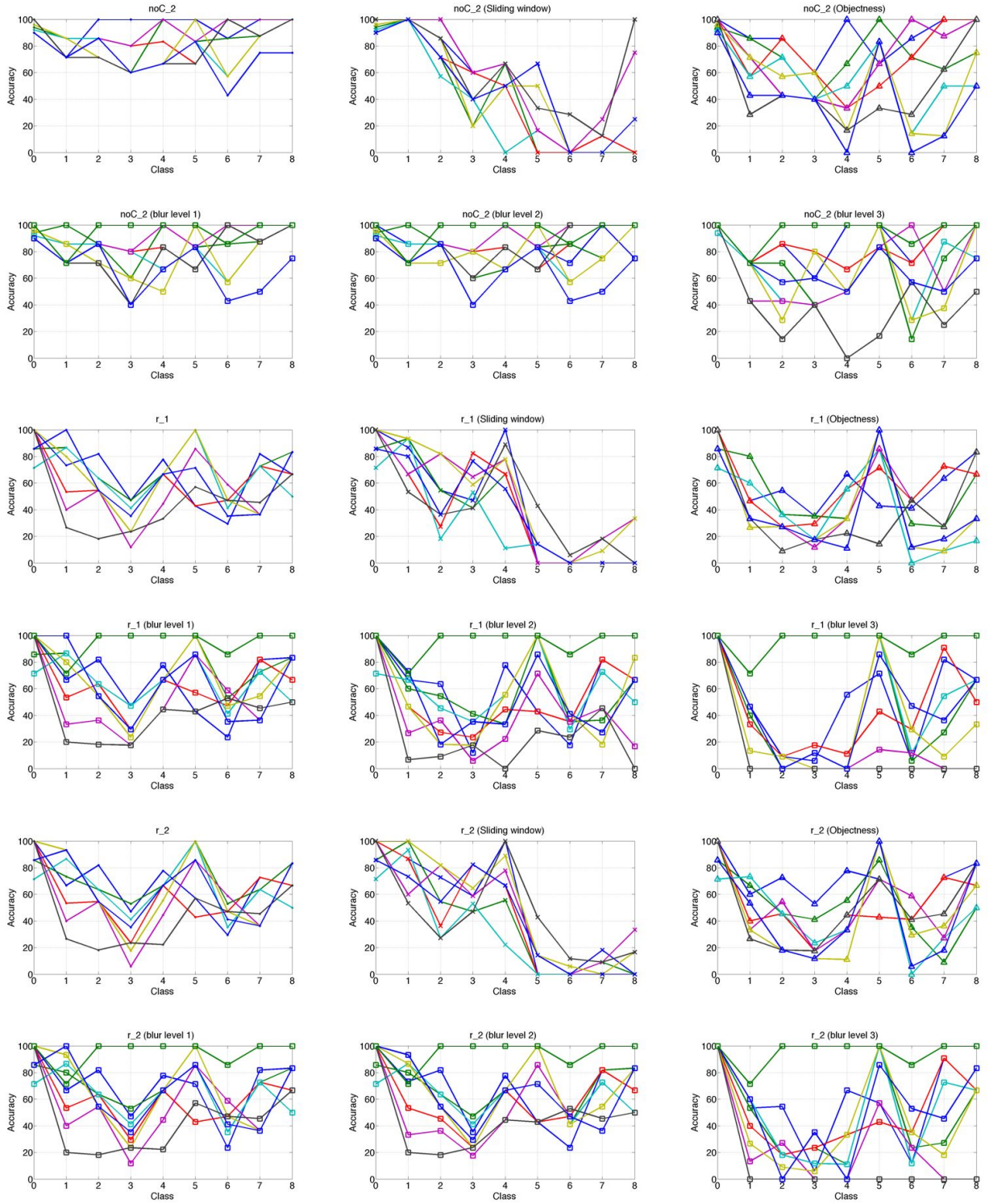

Figure A.4: For each dataset, accuracy (% of correct predictions) for each class (0-8). Each line represents the value computed for a different network.

A.3 F-measure

Figure A.5: For each dataset, F-measure for each class (0-8). Each line represents the value computed for a different network.

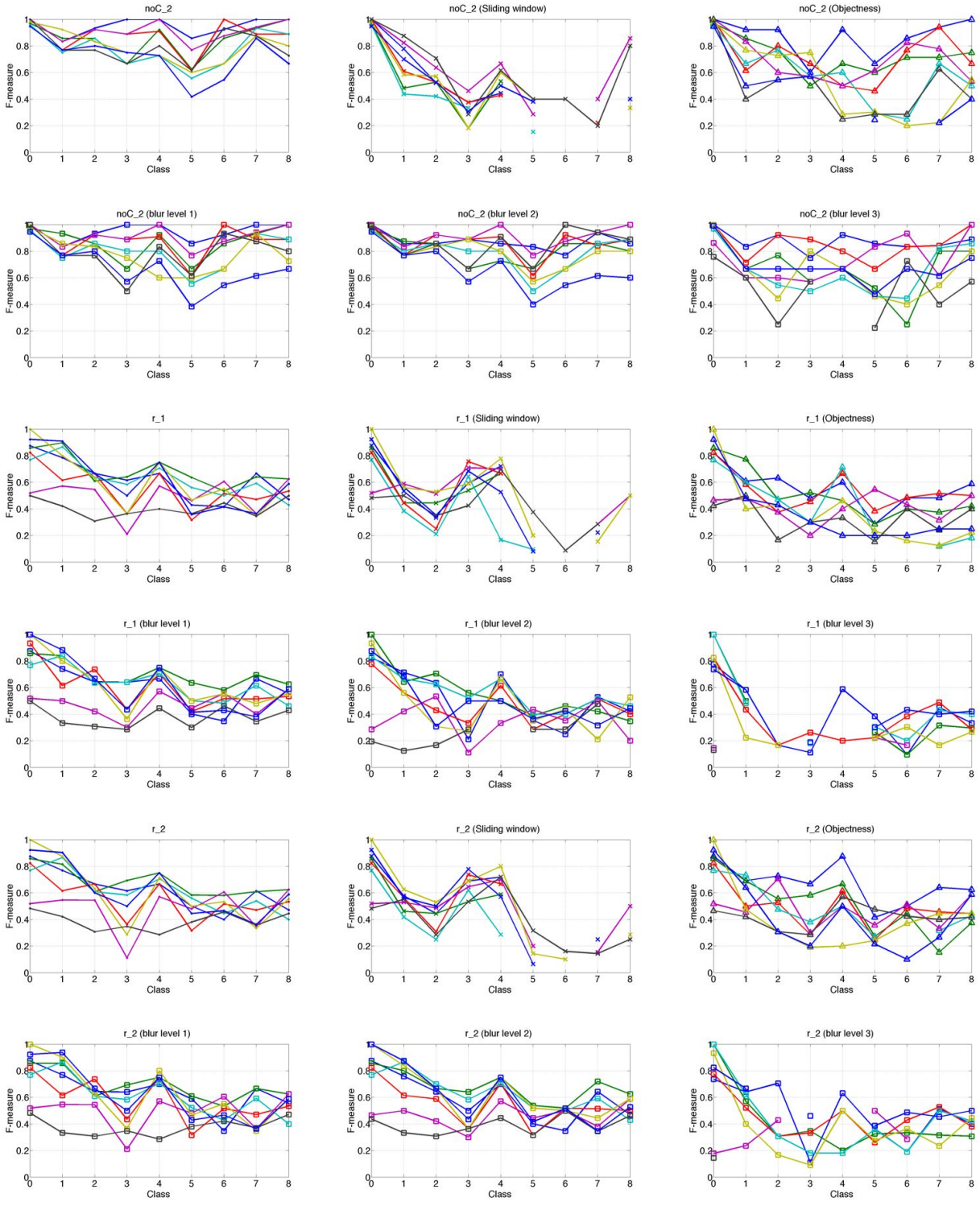

Figure A.6: For each dataset, F-measure for each class (0-8). Each line represents the value computed for a different network.

A.4 Confusion matrix

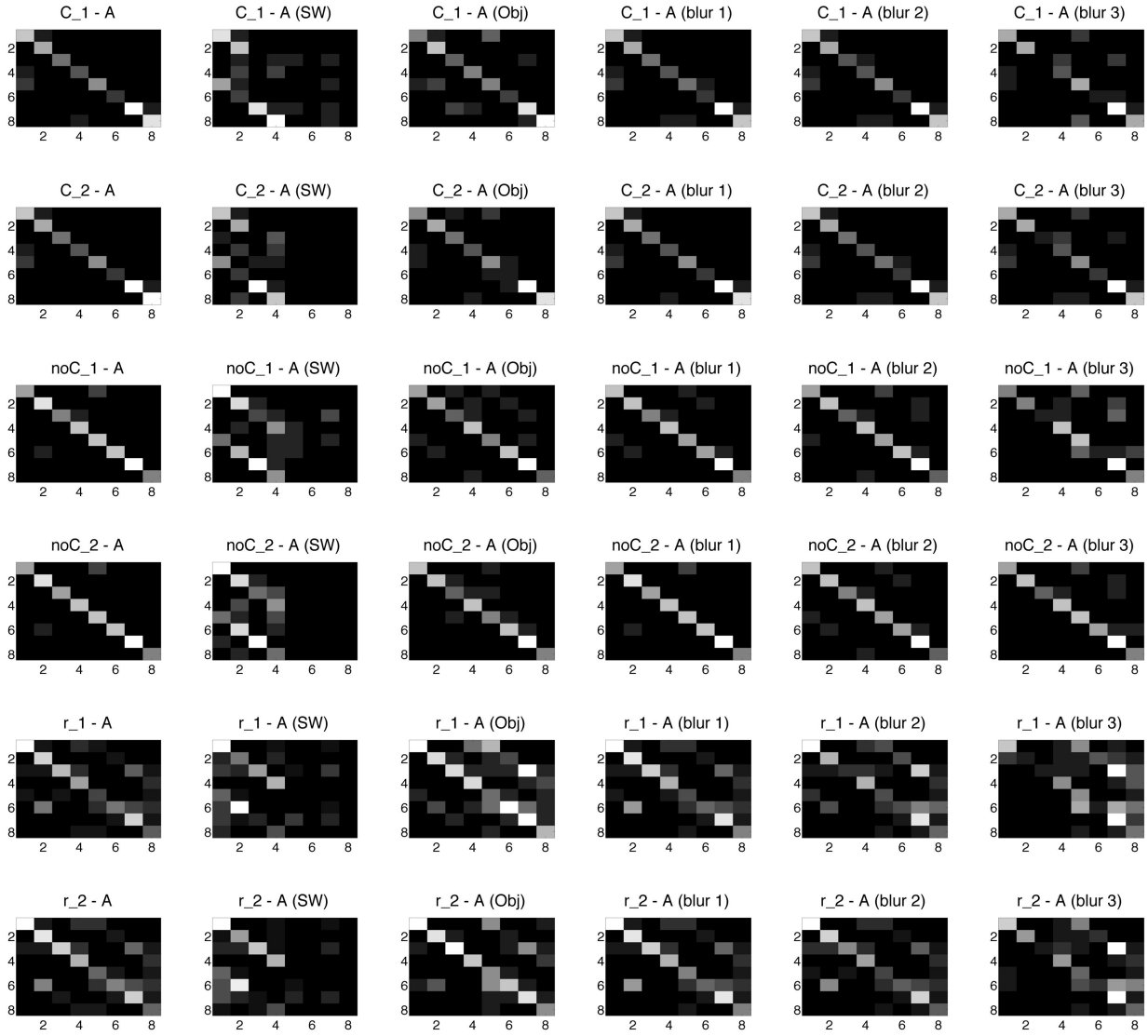

Figure A.7: For each dataset, confusion matrix for classes (1-8) for network A.

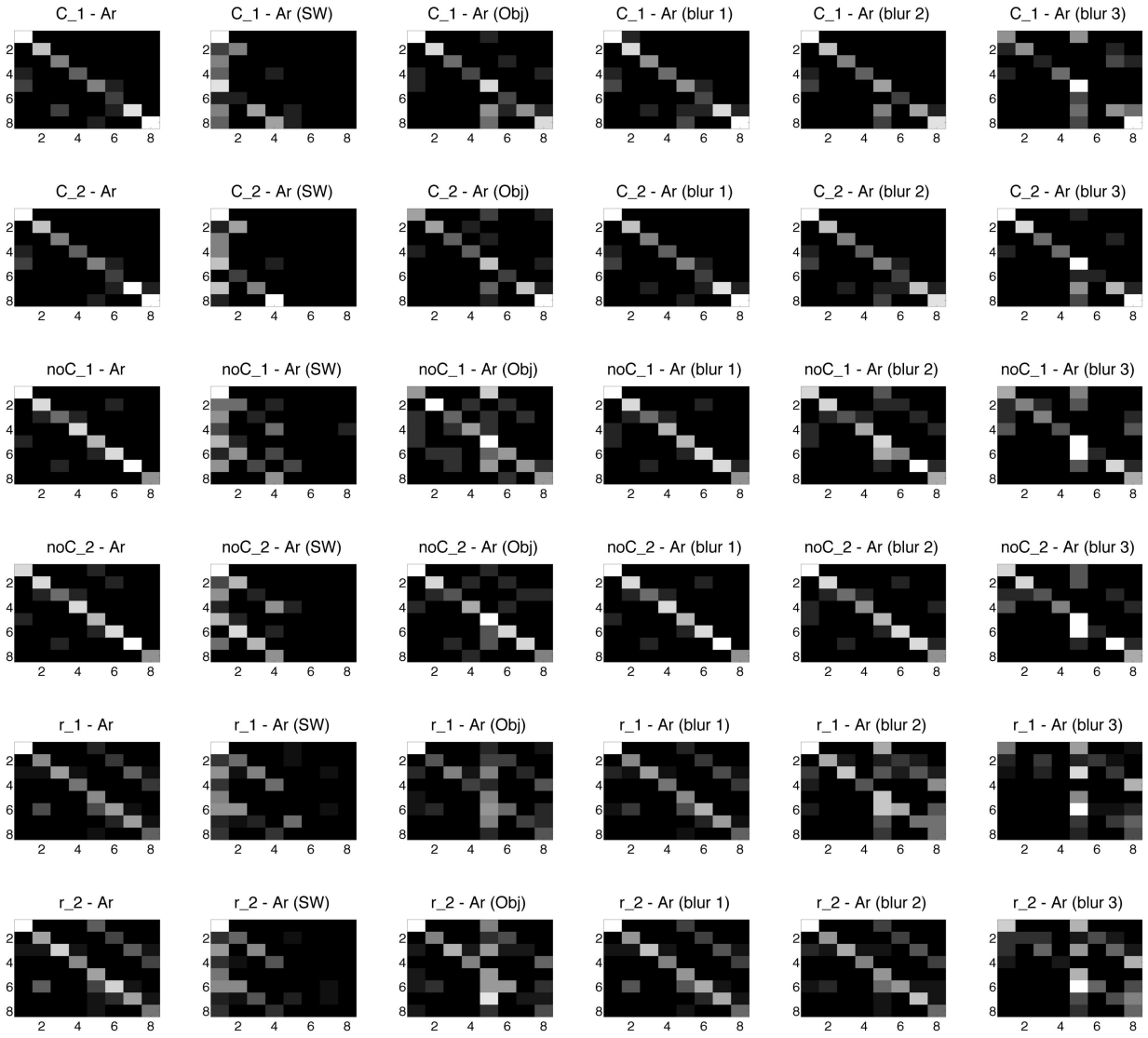

Figure A.8: For each dataset, confusion matrix for classes (1-8) for network Ar.

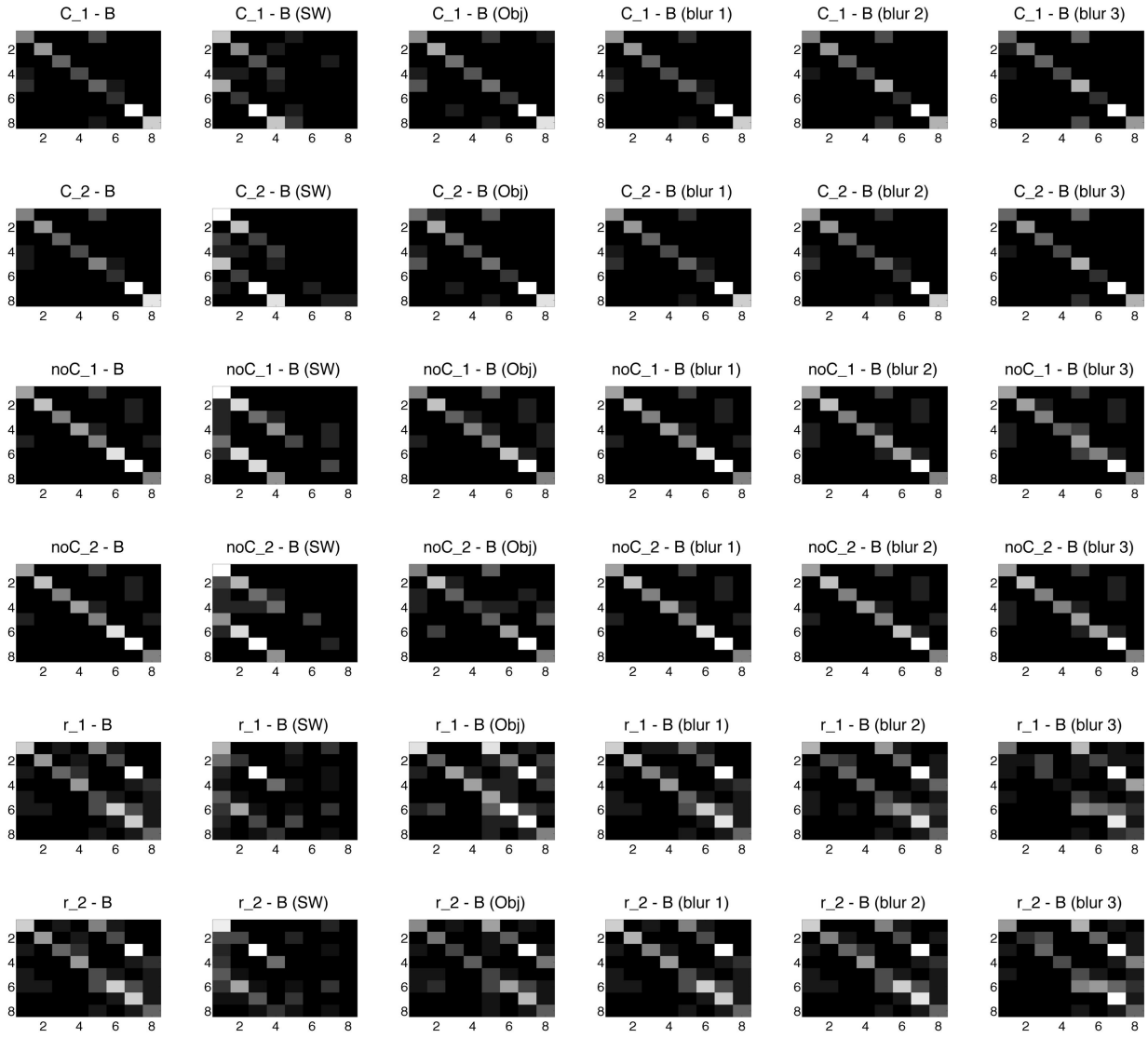

Figure A.9: For each dataset, confusion matrix for classes (1-8) for network B.

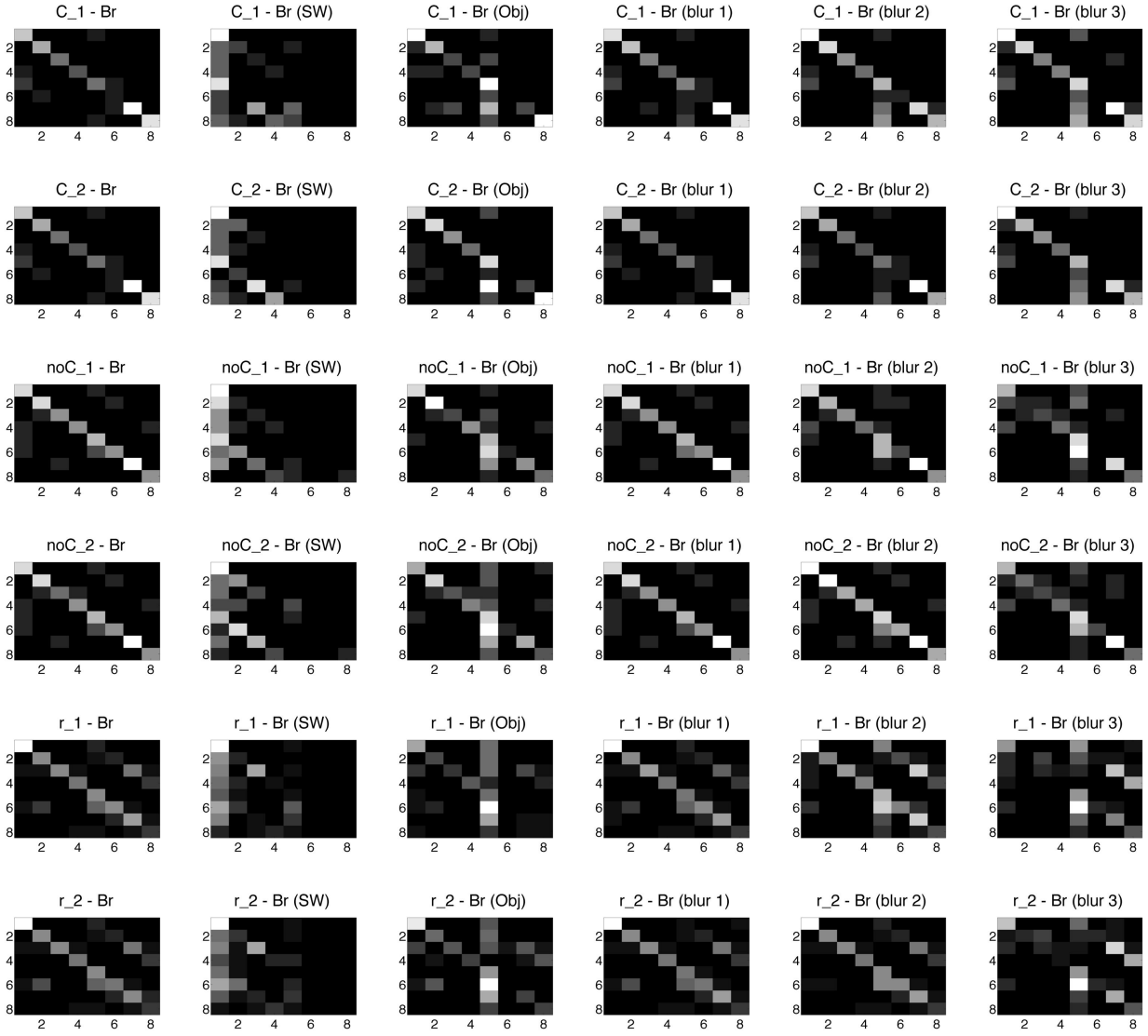

Figure A.10: For each dataset, confusion matrix for classes (1-8) for network Br.

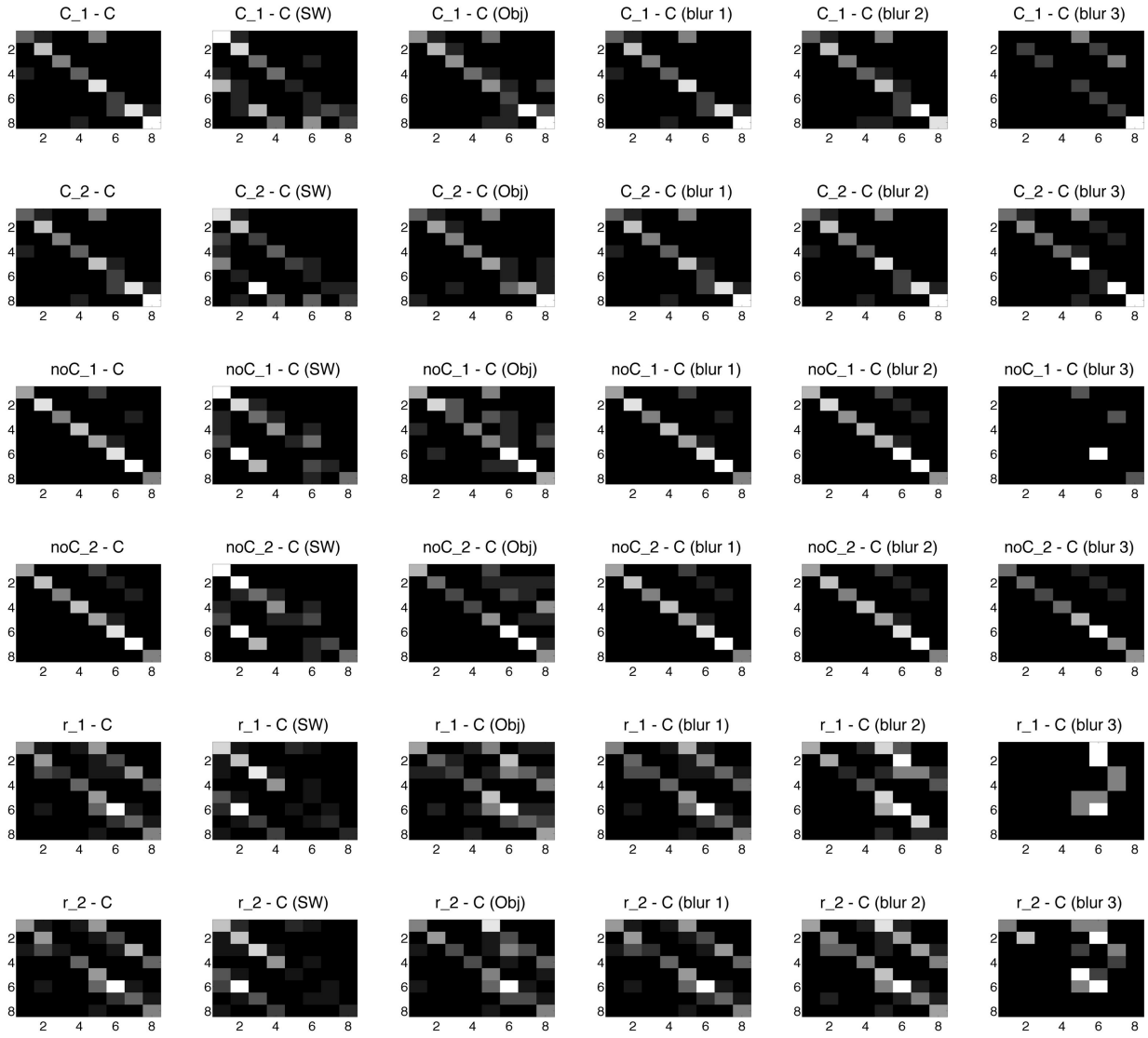

Figure A.11: For each dataset, confusion matrix for classes (1-8) for network C.

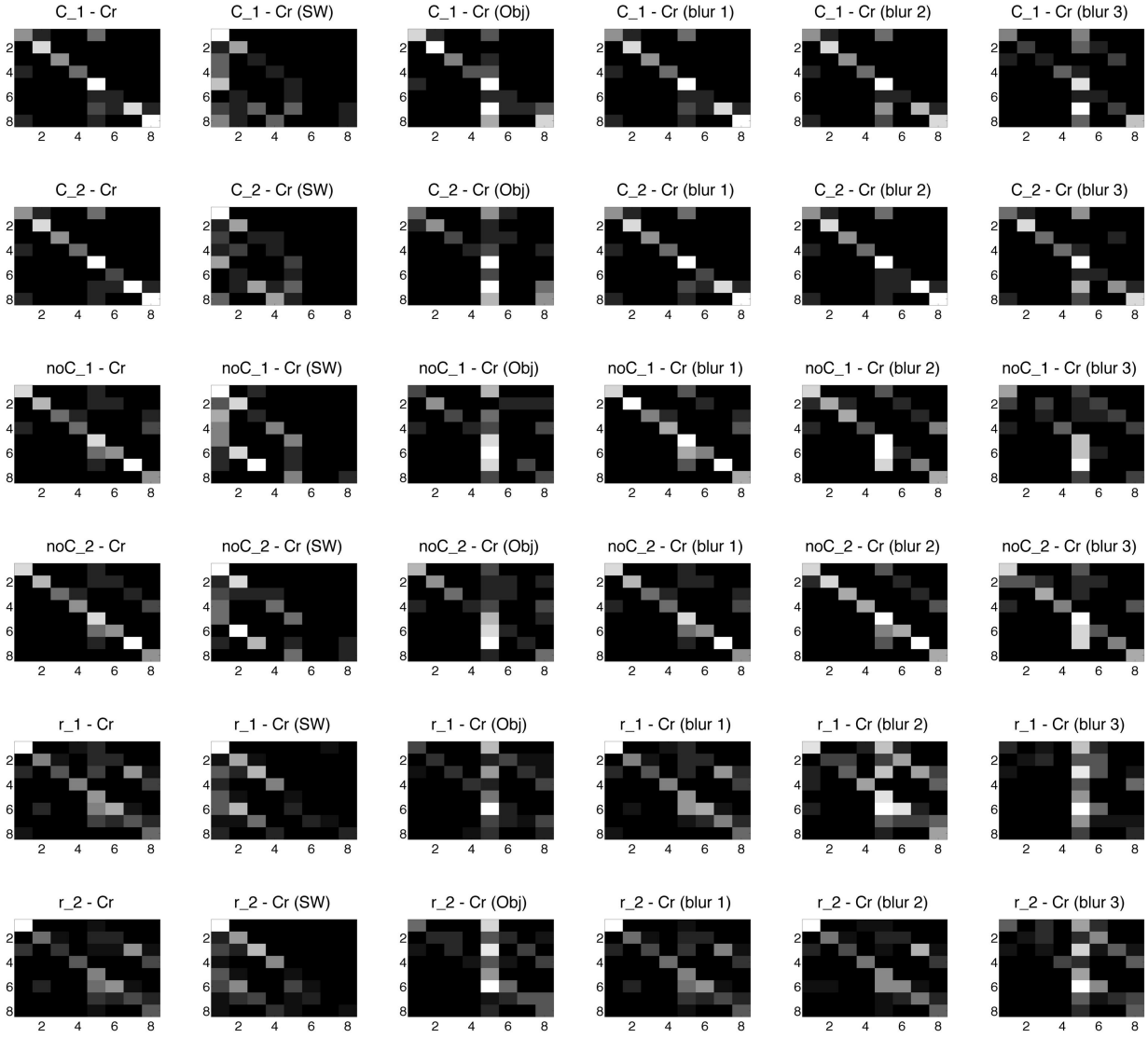

Figure A.12: For each dataset, confusion matrix for classes (1-8) for network Cr.

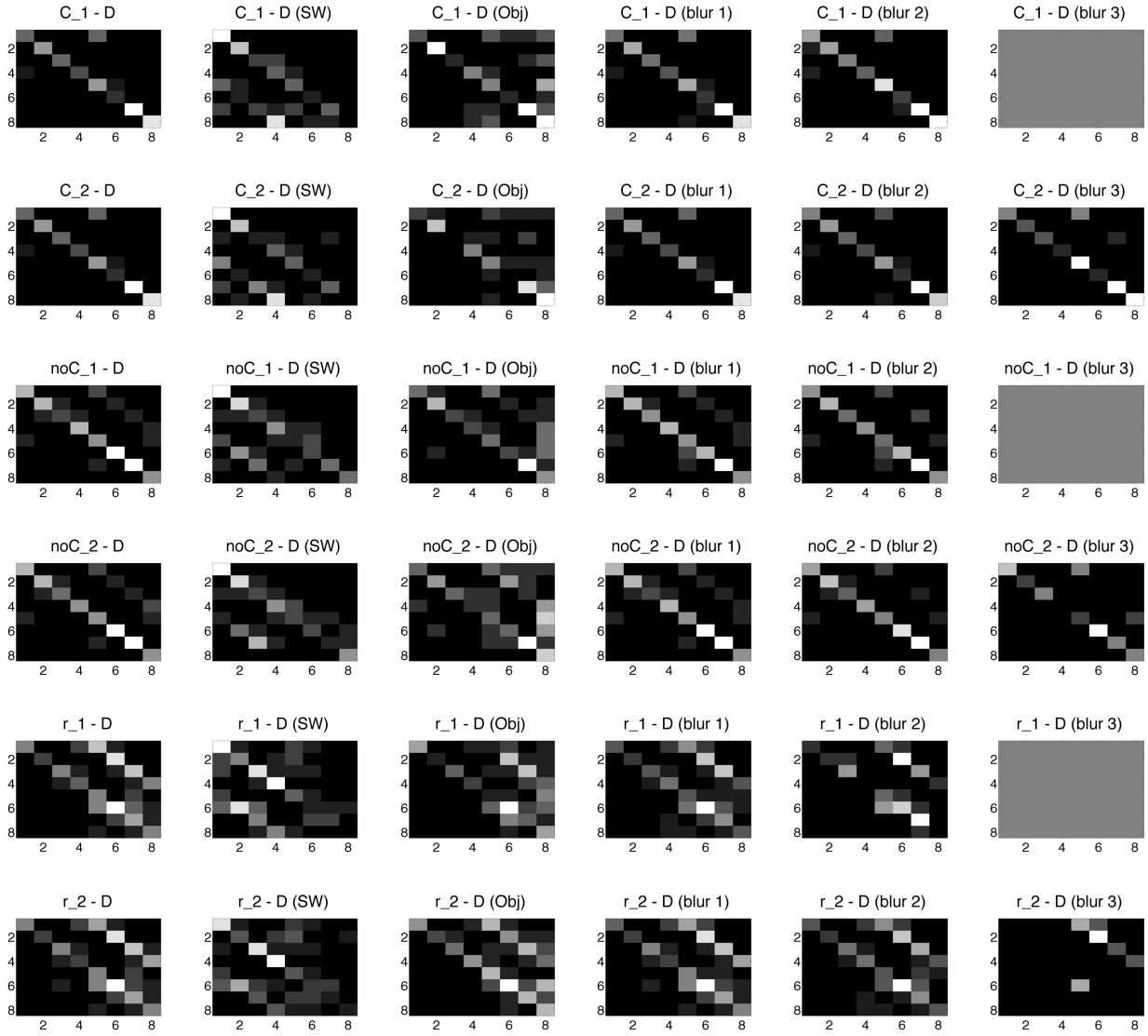

Figure A.13: For each dataset, confusion matrix for classes (1-8) for network D. Grey matrices indicate that the method failed to estimate the pose in that dataset.

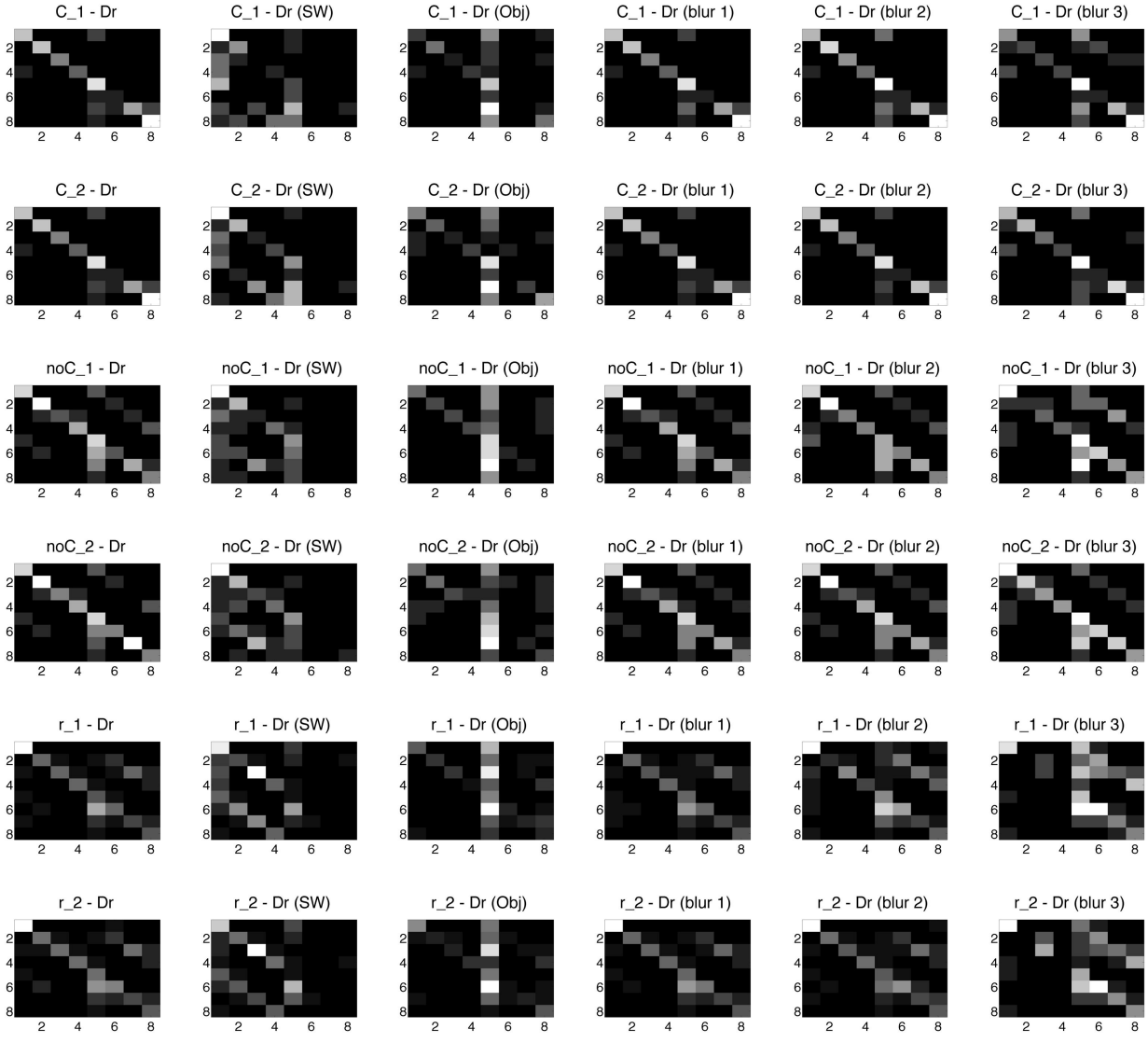

Figure A.14: For each dataset, confusion matrix for classes (1-8) for network Dr.

A.5 Time per image

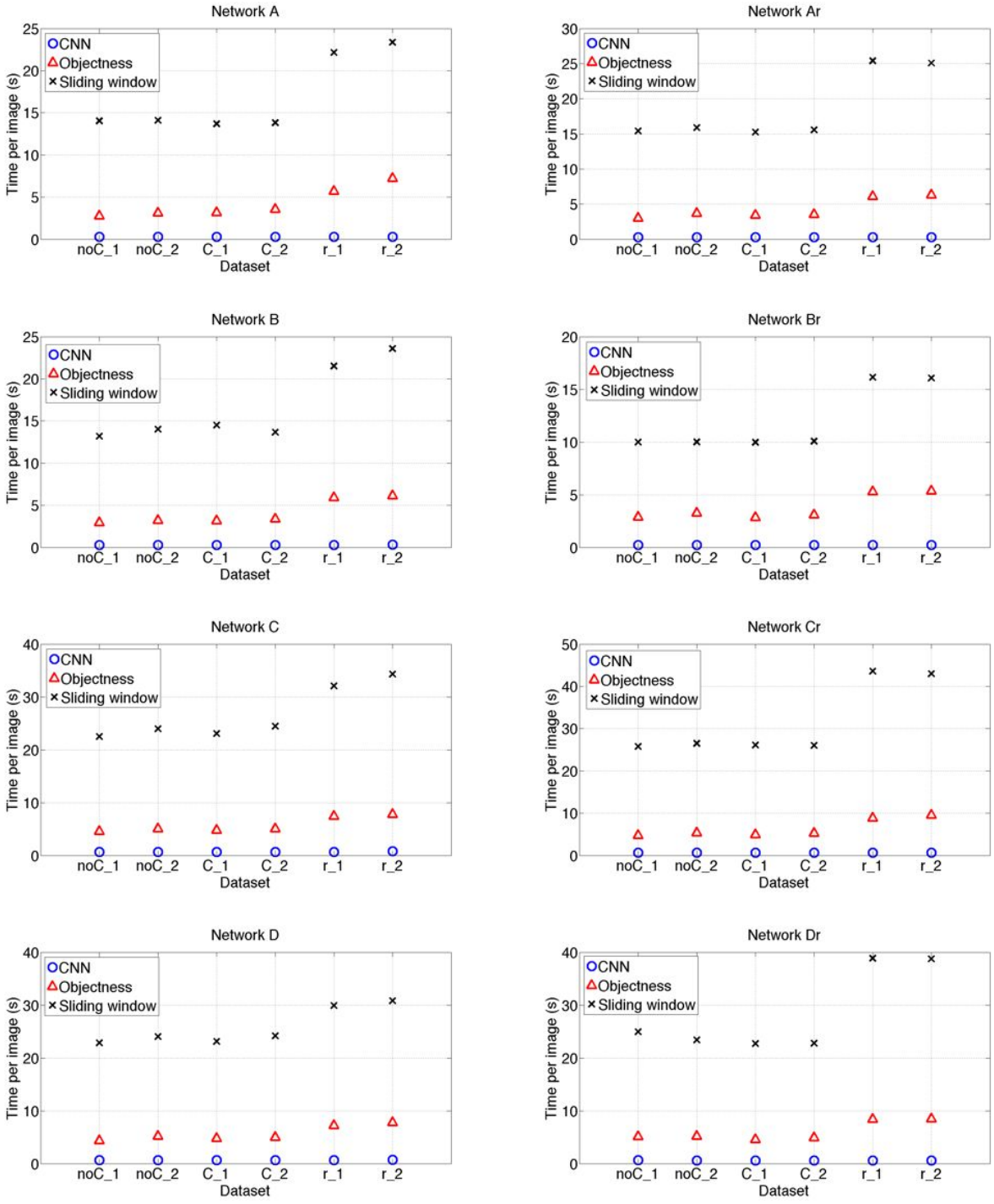

Figure A.15: Computational time per image for all the datasets, networks and methods.

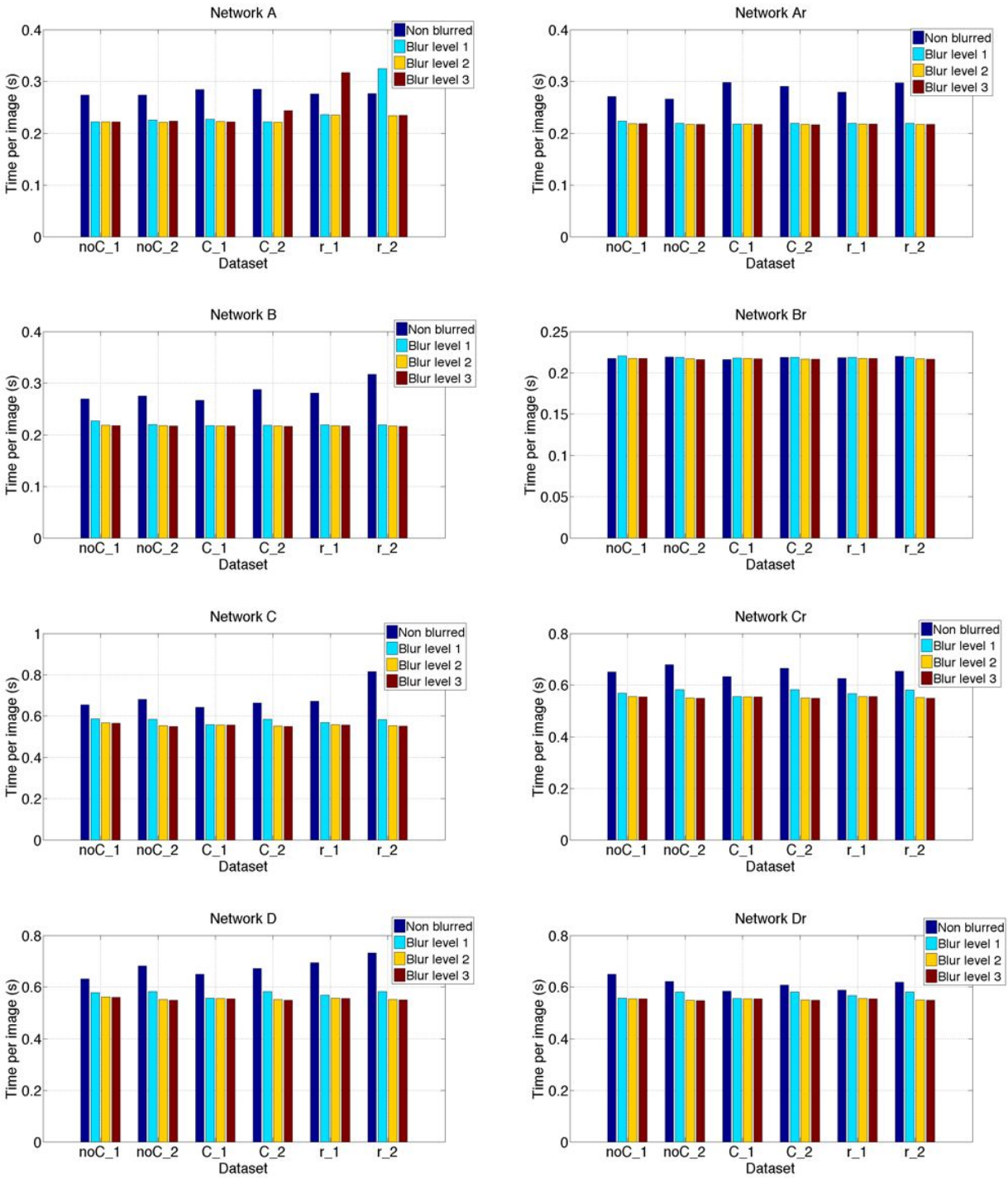

Figure A.16: Computational time per image for different levels of blurring of all the datasets and networks.

A.6 Results for RANSAC and Hager's PnP

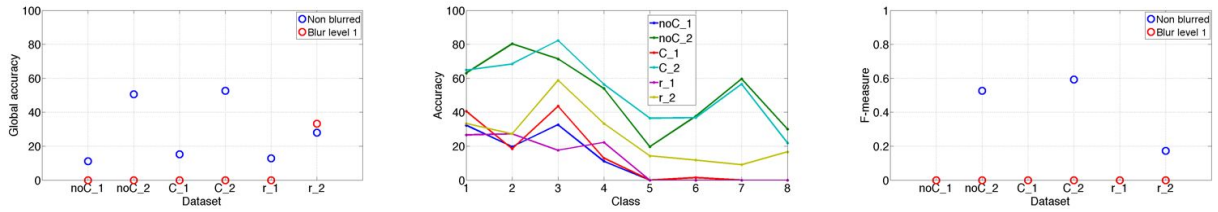

Figure A.17: From left to right: global accuracy for all datasets using RANSAC + Hager's PnP, per-class accuracy for classes (1-8) for all datasets and F-measure for all datasets..

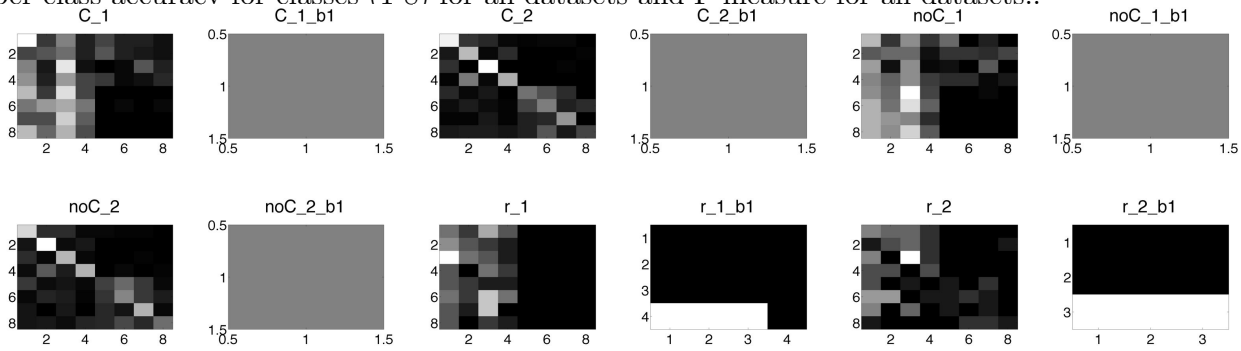

Figure A.18: Confusion matrix for classes (1-8) for all datasets.

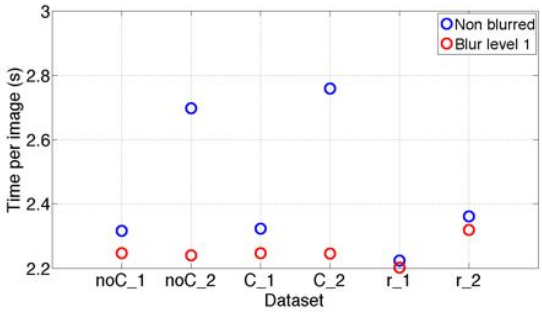

Figure A.19: Computational times of applying RANSAC + Hager's PnP to each dataset.